

Apple®

AppleCD SC™ Developers Guide®

Revised Edition

 APPLE COMPUTER, INC.

This manual is copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple Computer, Inc. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

The Apple logo is a registered trademark of Apple Computer, Inc. Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

© Apple Computer, Inc., 1989
20525 Mariani Avenue
Cupertino, CA 95014-6299
(408) 996-1010

Apple, the Apple logo, AppleShare, AppleTalk, Apple IIGS, MacApp, Macintosh, and ProDOS are registered trademarks of Apple Computer, Inc.

A/UX, APDA, AppleCD SC, APW, EtherTalk, Finder, GS/OS, HyperCard, HyperTalk, MPW, and MultiFinder are trademarks of Apple Computer, Inc.

IBM is a registered trademark of International Business Machines Corporation.

Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

LightSpeedC is a trademark of Symantec Corporation.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Linotronic is a registered trademark of Linotype Co.

POSTSCRIPT is a registered trademark, and Illustrator is a trademark, of Adobe Systems Incorporated.

Simultaneously published in the United States and Canada.

Contents

Figures and tables / viii

Preface About This Manual / xi

What this manual contains / xii

How to use this manual / xiii

Visual cues and conventions / xiv

Where to look for more information / xv

 Apple CD-ROM developer assistance / xvi

 Apple Developer Programs / xvi

 APDA / xvii

 Developing CD-ROM products for the retail market / xvii

1 Introduction to CD-ROM / 1

What is CD-ROM? / 2

Ideas for information distribution / 3

Why put information on CD-ROM? / 5

AppleCD SC features / 6

2 Creating a CD-ROM / 9

Product design / 11

Defining the delivery system / 14

 Software development / 15

Data preparation / 16

 Data collection / 17

 Data conversion / 17

 Data indexing / 18

Logical formatting / 19

Disc mastering / 20	
Tape premastering / 20	
Mastering the disc / 21	
Replication / 23	
Testing / 23	
Packaging / 24	
Equipment and software for product development / 24	
Developing software for a Macintosh / 24	
Developing software for the Apple II family / 26	
3 High Sierra/ISO 9660 and the AppleCD SC / 27	
The High Sierra and ISO 9660 file systems / 28	
ISO 9660 design goals / 29	
ISO 9660 file-naming conventions / 30	
ISO 9660 on the Macintosh / 31	
High Sierra and ISO 9660 discs on the Macintosh desktop / 33	
Additional information about ISO, Macintosh, and the AppleCD SC / 37	
High Sierra and ISO 9660 CD-ROM discs on the Apple IIGS / 38	
High Sierra and ISO 9660 CD-ROM discs on the Apple IIe / 38	
Comparing the High Sierra and ISO file systems with HFS / 39	
Apple extensions for ISO 9660 / 40	
Introduction / 40	
The Protocol Identifier / 41	
The Directory Record SystemUse field / 42	
The ProDOS filename transformation / 45	
Guidelines for transforming HFS filenames / 47	
ISO 9660 associated files / 47	
4 HyperCard and CD-ROM / 49	
A look at HyperCard for CD-ROM / 50	
HyperTalk: The HyperCard scripting language / 51	
HyperCard objects / 51	
HyperCard as a data-retrieval engine / 54	
Designing a CD-ROM retrieval engine with HyperCard / 54	
A friendly interface / 59	

Working with an existing CD-ROM /	61
The HyperCard interface for existing CD-ROM databases /	63
Additional interface considerations /	66
The ten general principles /	66
Programming /	68
Layout and design /	69
HyperCard interface design /	69
Testing your interface design /	69
Localization—international issues /	70
Limitations of the HyperCard search mechanism /	71
Increasing HyperCard's performance for CD-ROM /	71
Hints for creating HyperCard stacks for large databases /	72
HyperCard stacks on a CD-ROM and locked media /	72
Operations that can't be performed on CD-ROM /	73

5 AppleCD SC and Sound / 75

CD-Audio and processor sound for CD-ROM /	76
Using processor sound in applications /	78
Processor sound on the Macintosh and Apple II /	79
Using CD-Audio sound in applications /	80
Tips for developing a CD-ROM application with sound /	80
The build environment /	81
The test environment /	83
HyperCard and sound /	84
Storing sound for use with HyperCard /	85
Sound-capturing hardware /	86
Sound production software /	87
Additional software /	87
Controlling the AppleCD SC audio features /	88

6 Networks and the AppleCD SC / 105

Naming the CD-ROM server volume /	106
Updating the CD-ROM server volume /	107

7	AppleCD SC Software / 109
	What makes the AppleCD SC work? / 110
	Controlling the AppleCD SC driver from your application / 111
	Macintosh low-level routines / 112
	Macintosh Device Manager routines / 113
	Using the Macintosh AppleCD SC SCSI routines / 113
	Macintosh AppleCD SC driver control and status calls / 115
	Macintosh CD-ROM device driver control calls / 116
	Eject (7) / 116
	MediaIcon (21) / 117
	DriveIcon (22) / 117
	DriveInfo (23) / 118
	SCSI (77) / 118
	RSwitch (78) / 119
	VarBlk (79) / 119
	UserEject (80) / 120
	Macintosh CD-ROM device driver audio control calls / 121
	ReadTOC (100) / 127
	ReadQ (101) / 128
	ReadHead (102) / 129
	ATrkSearch (103) / 130
	APlay (104) / 131
	APause (105) / 132
	AStop (106) / 133
	AStatus (107) / 134
	AScan (108) / 135
	Macintosh CD-ROM device driver status calls / 136
	DiskStatus (8) / 136
	WhoIsThere (97) / 137
	GetSize (98) / 137
	GetSense (99) / 138
	Apple II AppleCD SC SmartPort control call descriptions / 139
	The SmartPort interface / 139

AppleCD SC SmartPort control calls /	142
Eject (\$04) /	142
TestUnitReady (\$05) /	143
RequestSense (\$06) /	143
ModeSelect (\$08) /	145
ModeSense (\$09) /	162
ReadCapacity (\$0D) /	167
Inquiry (\$0F) /	168
SetBlockSize (\$13) /	171
SetTimeout (\$14) /	172
Seek (\$16) /	172
StartUnit/StopUnit (\$18/\$19) /	173
PreventRemoval/AllowRemoval (\$1A/\$1B) /	173
Verify (\$1C) /	174
RezeroUnit (\$1D) /	175
Patch1Call (\$1E) /	175
SetNewSDAT (\$1F) /	176
AppleCD SC SmartPort audio control calls /	177
AudioSearch (\$20) /	177
AudioPlay (\$21) /	181
AudioPause (\$22) /	184
AudioStop (\$23) /	185
AudioStatus (\$24) /	187
AudioScan (\$25) /	189
ReadTOC (\$27) /	192
ReadQSubcode (\$28) /	196
ReadHeader (\$29) /	198

A Sense Code Definitions / 201

Glossary / 205

Index / 209

Figures and tables

CHAPTER 1 Introduction to CD-ROM / 1

Figure 1-1 AppleCD SC features / 6

CHAPTER 2 Creating a CD-ROM / 9

Figure 2-1 The steps to create a CD-ROM / 10

Figure 2-2 The steps involved in the CD-ROM mastering process / 22

CHAPTER 3 High Sierra/ISO 9660 and the AppleCD SC / 27

Figure 3-1 The CD-ROM software environment / 31

Figure 3-2 An Eject/Initialize dialog box / 33

Figure 3-3 Initialization failed dialog box / 33

Figure 3-4 The CD-ROM icon on the desktop / 34

Figure 3-5 Viewing the contents of a High Sierra disc / 35

Figure 3-6 Viewing the contents of a directory on a High Sierra disc / 36

Figure 3-7 File size shown in the Get Info dialog box and CD-ROM volume window / 37

Table 3-1 CD-ROM data organization comparison / 39

Table 3-2 SystemUseID / 43

Table 3-3 SystemUseID 01, ProDOS / 43

Table 3-4 SystemUseID 02, HFS / 43

Table 3-5 SystemUseID 03, HFS: bundle bit set / 44

Table 3-6 SystemUseID 04, HFS / 44

Table 3-7 SystemUseID 05, HFS: bundle bit set / 44

Table 3-8 SystemUseID 06, HFS: bundle bit set / 45

Table 3-9 ProDOS-to-ISO 9660 filename transformation / 46

CHAPTER 4 HyperCard and CD-ROM / 49

- Figure 4-1 HyperCard objects / 52
- Figure 4-2 The layers of HyperCard objects on cards and backgrounds / 53
- Figure 4-3 Pieces of a HyperCard-based CD-ROM (data preparation) / 58
- Figure 4-4 Pieces of a HyperCard-based CD-ROM (data playback) / 59
- Figure 4-5 Example of HyperCard interface / 60
- Figure 4-6 Example of graphic elements that enhance the data presentation / 61
- Figure 4-7 Preparing an existing non-Macintosh CD-ROM for the Macintosh environment / 62
- Figure 4-8 Sample card for text search / 63
- Figure 4-9 Book titles corresponding to author's name / 64
- Figure 4-10 Detailed information in catalog card / 65

CHAPTER 5 AppleCD SC and Sound / 75

- Figure 5-1 A sample CD-ROM master tape listing / 82
- Table 5-1 Comparison of processor and CD-Audio sound / 77

CHAPTER 7 AppleCD SC Software / 109

- Figure 7-1 Device software environment / 111
- Figure 7-2 CD-ROM data organization / 122
- Figure 7-3 Request Sense return sense data format / 144
- Figure 7-4 Mode Select header data format / 146
- Figure 7-5 Mode Select block descriptor format / 147
- Figure 7-6 Mode Select page descriptor format / 148
- Figure 7-7 Mode Select unit attention page format / 149
- Figure 7-8 Mode Select error recovery page format / 151
- Figure 7-9 Mode Select disconnect/reconnect page format / 158
- Figure 7-10 Mode Select drive parameters page format / 161
- Figure 7-11 Mode Sense command format / 162
- Figure 7-12 Mode Sense header data format / 164
- Figure 7-13 Mode Sense block descriptor data format / 165
- Figure 7-14 Mode Sense page descriptor data format / 166
- Figure 7-15 Read Capacity return data format / 168
- Figure 7-16 Inquiry command return data format / 169
- Figure 7-17 AudioSearch address in type \$00 logical block format / 180

Figure 7-18	AudioSearch address in type \$01 absolute minutes-seconds-frames format / 180
Figure 7-19	AudioSearch address in type \$02 track number format / 181
Figure 7-20	AudioStatus data format / 188
Figure 7-21	Scan start address type \$00 logical block format / 191
Figure 7-22	Scan start address type \$01 in absolute minutes-seconds-frames format / 191
Figure 7-23	Scan start address type \$02 track number format / 192
Figure 7-24	ReadTOC type \$00 data format / 194
Figure 7-25	ReadTOC type \$01 data format / 195
Figure 7-26	ReadTOC type \$02 data format / 195
Figure 7-27	Q Subcode data format / 197
Figure 7-28	Header data format / 199
Table 7-1	Description of the signed value in the cspTransSize field / 114
Table 7-2	AppleCD SC SmartPort control call codes / 141
Table 7-3	Apple Common Command Set page codes / 149
Table 7-4	Valid error recovery page byte values for the AppleCD SC modes of operation / 154
Table 7-5	CD-ROM error recovery procedures / 155
Table 7-6	Relationships between buffer full ratio and number of blocks / 159
Table 7-7	PCF bit values and definitions / 163
Table 7-8	Response data format codes / 170
Table 7-9	Values for search address type parameter / 179
Table 7-10	Values for playback address type parameter / 183
Table 7-11	Values for stop address type parameter / 186
Table 7-12	AudioStatus data field byte values and definitions / 188
Table 7-13	Control field bit values and definitions / 189
Table 7-14	Values for scan start address type parameter / 190
Table 7-15	Values for ReadTOC type parameter / 194
Table 7-16	Control field bit values and definitions / 198

APPENDIX A Sense Code Definitions / 201

Table A-1	Additional Sense codes / 202
-----------	------------------------------

Preface **About This Manual**

The *AppleCD SC Developer's Guide* provides information for information providers, experienced computer users, system administrators, and application programmers who want to take advantage of the features of the AppleCD SC™ drive. To use this manual, you should have some knowledge of a programming language, such as BASIC, Pascal, C, or assembly language, and you should be familiar with the computer for which you intend to develop AppleCD SC software.

You do *not* need to use this manual when simply running packaged programs for your Apple® computer. However, this manual can help you if you are running a program that can be configured to take advantage of the features of the AppleCD SC, or if you are writing or modifying a program that uses the AppleCD SC.

Instructions for connecting the AppleCD SC to your computer, inserting CD-ROM cartridges, and other routine operating tasks are provided in the *AppleCD SC Owner's Guide* that came with your AppleCD SC.

This preface describes the contents of this manual and the visual cues and conventions used throughout the book. It also lists some other books that you may wish to refer to when using this manual. ■

What this manual contains

The *AppleCD SC Developer's Guide* describes the operating modes and special capabilities of the AppleCD SC as well as the software interface to the host computer.

Here is a brief outline of the contents of this manual:

- Chapter 1, "Introduction to CD-ROM," includes a description of the features and possible uses for AppleCD SC.
- Chapter 2, "Creating a CD-ROM," steps through the preparation process for designing, collecting, and preparing information to be placed on CD-ROM discs.
- Chapter 3, "High Sierra/ISO 9660 and the AppleCD SC," includes an overview of the Macintosh® system software for reading CD-ROM discs with High Sierra and ISO 9660 file system formats. It also describes how High Sierra and ISO 9660 differ from the native HFS and ProDOS® interfaces on the Macintosh and Apple II.
- Chapter 4, "HyperCard and CD-ROM," describes considerations for choosing a retrieval model. It explains the advantages of using HyperCard® as the data retrieval engine or as an interface layer between a custom retrieval engine and large databases. This chapter provides examples of HyperCard retrieval interfaces, and describes how one can utilize HyperCard external commands to bring existing database libraries into the Macintosh environment.
- Chapter 5, "AppleCD SC and Sound," provides an introduction to getting sound into the Macintosh software environment and a program example of how to implement the CD-Audio capability on the AppleCD SC.
- Chapter 6, "Networks and the AppleCD SC," provides implementation-specific information for using the AppleCD SC as a server on a network.
- Chapter 7, "AppleCD SC Software," provides a functional description of how the Macintosh and Apple II software works at the operating system level to allow the user to read data from the AppleCD SC. It also includes a description of the Macintosh AppleCD SC control calls and Apple II AppleCD SC SmartPort control calls.

This book also contains a glossary of technical terms used in the text and an index.

How to use this manual

This manual provides information for programmers and developers with a variety of needs and levels of experience. The following guidelines can help you get the most out of this book.

- If you are not writing programs but are interested in learning more about CD-ROM in general and about the AppleCD SC in particular, read Chapters 1 and 2. Look through the rest of the book to get an idea of the kinds of tasks the AppleCD SC can do. You might also want to read Chapters 3, 4, and 7 to learn about the considerations involved in writing programs for your AppleCD SC.
- If you are using this book to learn how to configure or customize a program to take advantage of the AppleCD SC's features, read Chapters 3, 4, 5, and 7.
- If you have experience writing programs that use CD-ROM but are unfamiliar with the AppleCD SC, look through Chapters 1 through 5 to learn about the features and options available. Then go back to the detailed description of a command in Chapter 7 when you are ready to use it.
- If you are a developer working on ISO 9660, a developer of CD-ROM authoring tools, or a publisher of ISO 9660 CD-ROM discs, read the section "Apple Extensions for ISO 9660" in Chapter 3.

Visual cues and conventions

Look for these visual cues throughout the manual:

◆ *Note:* Notes like this contain supplementary information.

▲ **Warning** Warnings like this direct your attention to something that could damage either software or hardware or result in loss of data. ▲

Special terms are shown in **boldface** type the first time they appear. The glossary contains definitions of these terms and other related technical terms.

A special typeface is used for characters that you type or for lines of program code:

It looks like this.

Hexadecimal numbers are preceded by a dollar sign (\$), except in tables where hexadecimal numbers are clearly labeled. For example, the hexadecimal equivalent of decimal 16 is written as \$10.

Where to look for more information

The *AppleCD SC Developer's Guide* assumes that you are familiar with CD-ROM technology and many of the possible uses for CD-ROM. However, additional information and ideas are presented in these documents:

- The *AppleCD SC Owner's Guide*, which is shipped with every AppleCD SC, explains how to set up an AppleCD SC in the standard configuration. The manual gives basic operating information, such as how to insert, eject, and store discs.
- *Human Interface Guidelines: The Apple Desktop Interface*, published by Addison-Wesley.
- *The HyperCard Script Language Guide: The HyperTalk Language*, published by Addison-Wesley.
- *CD ROM: The New Papyrus*, published by Microsoft Press.
- *CD ROM 2: Optical Publishing*, published by Microsoft Press.
- *CD ROM 3: Interactive Multimedia*, foreword by John Sculley, published by Microsoft Press, co-published with Apple Computer, Inc.
- *Inside Macintosh*, published by Addison-Wesley.
- *Programmer's Introduction to the Macintosh Family*, published by Addison-Wesley.
- *XCMD's for HyperCard*, published by the MIS: Press.

In addition to the *HyperCard Script Language Guide* mentioned above, several books about HyperCard and HyperTalk™ for the Macintosh are available from book stores.

Apple CD-ROM developer assistance

Apple Developer Technical Support provides technical information and support for Apple certified software developers. If you are a certified developer and have questions regarding CD-ROM, send your questions to AppleLink address MACDTS or to MCI mail address MACDTS. Please title your inquiry *AppleCD SC* so that your questions are directed to the proper group. If your questions are actually more specific to HyperCard than to CD-ROM, title your inquiry *HyperCard 1.2.1*.

Apple Developer Programs

If you plan to develop hardware or software products for sale through retail channels, you can get valuable support from Apple Developer Programs. Write to

Apple Developer Programs
Apple Computer, Inc.
20525 Mariani Avenue, MS 51-W
Cupertino, CA 95014-6299
408-974-4897

Apple Developer Programs also distributes Macintosh and Apple II Technical Notes containing current information important to programmers.

APDA

APDA™ provides a wide range of technical products and documentation, from Apple and other suppliers, for programmers and developers who work on Apple equipment. For information about APDA, contact

APDA
Apple Computer, Inc.
20525 Mariani Avenue, MS 33-G
Cupertino, CA 95014-6299
1-800-282-APDA (1-800-282-2732)
Fax: 408-562-3971
Telex: 171-576
AppleLink: APDA

Developing CD-ROM products for the retail market

If you are preparing a CD-ROM product for sale in the retail market, you should apply for an ISBN (International Standard Book Number). The ISBN provides a unique reference number for items being sold in bookstores in any country. Bookstores order products from databases by referring to the product ISBN.

If your CD-ROM product will be updated on a regular basis, you should apply for an ISSN (International Standard Serial Number) instead of an ISBN.

You can request a free application for an ISBN by contacting Bowker Publishers in New York at 212-337-6975. If you need an ISSN, contact the Library of Congress at 202-287-6452. The application for an ISSN is also free of charge.

Chapter 1 Introduction to CD-ROM

The *AppleCD SC Developer's Guide* is primarily for the professional developer or information provider interested in working on CD-ROM products for use with Macintosh or Apple II computers. The *AppleCD SC Developer's Guide* also provides basic information for anyone interested in knowing about CD-ROM. To satisfy these diverse audiences, this introduction contains basic information about CD-ROM and the AppleCD SC drive. Things you need to know before you decide to develop software or put information on CD-ROM to be used with the Macintosh and Apple II are contained in Chapter 2.

If you already understand CD-ROM and the steps involved in the CD-ROM creation process, but are interested in learning about the AppleCD SC device software, you may want to skip ahead and start your reading at Chapter 3. ■

What is CD-ROM?

CD-ROM (*compact disc read-only memory*) is an optical distribution storage and retrieval medium for large amounts of digital data. CD-ROM is a relatively new technology, which stems directly from the audio-CD industry. CD-ROM data is stored on a hard plastic disc, measuring 12 centimeters (4 $\frac{3}{4}$ inches) in diameter and slightly over one millimeter thick ($\frac{1}{20}$ inch).

A CD-ROM can store 656 megabytes (MB) in **mode 1**, or up to 748 megabytes in **mode 2**. You could compare that to about 270,000 single-spaced typewritten pages, or the contents of 820 floppy disks with a capacity of 800 kilobytes (K) each. Data on a CD-ROM is not limited strictly to text and graphics. Anything that can be digitized for use on the computer can be stored on CD-ROM discs. Examples of commonly digitized data include: computer application software, voice, music, and graphics.

The popularity of CD-ROM is based not only on its high capacity for data storage, but also on other admirable qualities. Here are some examples:

- Information on CD-ROM is stored as read-only memory, which means it cannot be changed or deleted.
- CD-ROM provides immediate access to large amounts of information, and allows easy, rapid cross-referencing of information.
- The discs are constructed of high-impact polycarbonate and are far less susceptible to damage by scratches, dust, magnetic fields, or water than floppy disks are. Additionally, when placed in the AppleCD SC caddy, a disc is basically free from any danger of mishandling in the typical office environment.
- CD-ROM mastering and reproduction costs are quite low on a per megabyte basis, and continue to decrease as CD-ROM technology matures.
- It is possible to simulate a CD-ROM's performance characteristics before actually producing the disc, thereby reducing some of the cost of product development.

Given that a CD-ROM disc is small, is nearly indestructible, and has the capacity for storing large amounts of data, it is an ideal way of getting information into the personal computer user's hands. The next section describes some of the possible uses for CD-ROM and the AppleCD SC.

Ideas for information distribution

CD-ROM is an excellent vehicle for widely dispersing extremely large databases of information. Here are some examples of the areas in which CD-ROM would be most useful:

- **Art and publications**

- graphic image databases
- dictionaries, thesauruses, and style guides
- design element, glossary, and document template databases

- **Education**

- interactive tutorials
- multilingual audiovisual dictionaries and encyclopedias

- **Industry**

- manufacturer parts lists, which break down unit assemblies into components lists
- component and product specifications
- engineering and mechanical drawings
- on-line technical documentation

- **Government and legal**

- U.S. federal trademark information with graphics
- agency regulations
- court decisions
- census data
- maps at national, state, regional, and metropolitan levels

- **Finance**

- financial statements
- U.S. federal tax forms
- market research and analysis reports
- stock market history

- **Science and medicine**
 - cancer research data
 - medical dictionary
 - scientific writings
 - toxicology database
- **Reference**
 - encyclopedias
 - library catalogs
 - periodicals and newspapers
- **Entertainment**
 - games
 - narrated childrens books
 - public domain software

As you can see, there are plenty of areas to explore, and this list only touches on the possibilities. Later, in Chapter 4, an example of a possible Macintosh interface for a CD-ROM educational reference is presented in the section "HyperCard as a Data-Retrieval Engine."

Why put information on CD-ROM?

Let's consider the advantages of the CD-ROM over conventional reference media, such as books, on-line services, or floppy disks, any one of which might supply the same types of information as does CD-ROM.

One of the advantages of CD-ROM is that it doesn't take up as much space as the other reference media. For example, an entire collection of encyclopedias on your desk would be hard to handle and cumbersome to search through. On the other hand, an AppleCD SC with an entire encyclopedia on CD-ROM could fit under your computer, or be placed away from your work area, and with the right application software the AppleCD SC would supply information with just a click of the mouse. Also, because it's easy to cut and paste information with Apple computers, this diverse amount of information can be easily integrated into your productive environment. You would no longer need to struggle with a large stack of books.

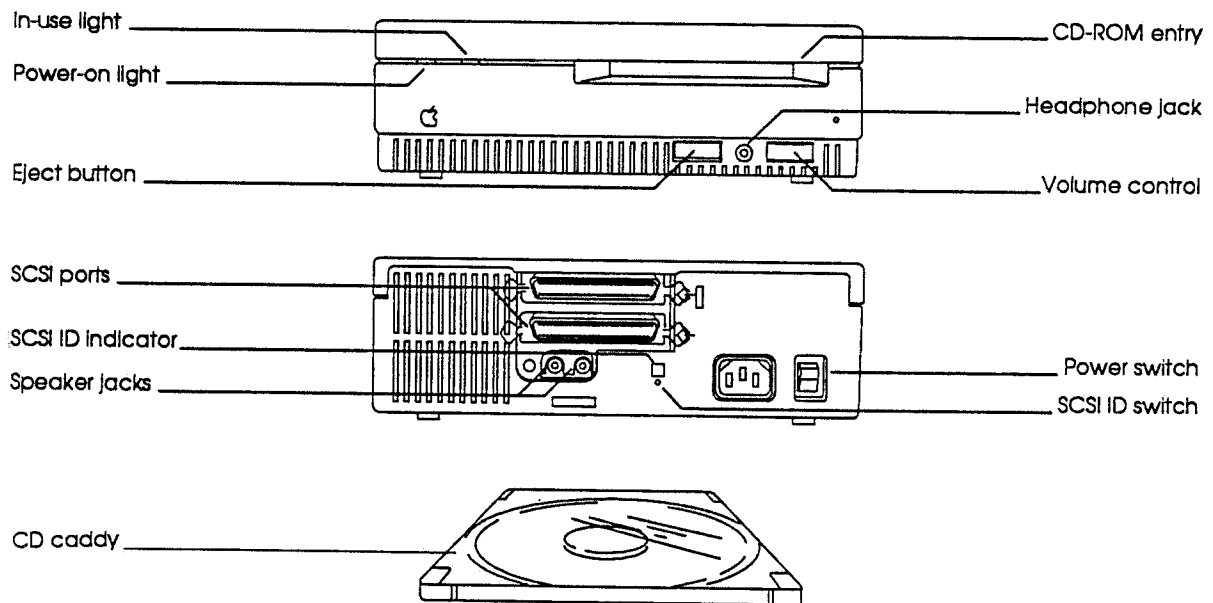
In an office where many people might use on-line services to access information, CD-ROM can be an inexpensive and more efficient alternative. On-line services run on telephone lines, which cannot provide the data transfer rates of an AppleCD SC connected directly to your computer. Slow transfer rates over telephone lines mean that searching for information and moving it into the computer memory for use will be a comparatively slow, frustrating process. Additionally, connection time can be very expensive. Having the ability to quickly search and transfer data from the CD-ROM connected to the computer will make users happier, and save on the operating overhead created by excessive use of telephone service.

We've already mentioned the advantages of CD-ROM over floppy disks—CD-ROM holds far more information, it's not easily destroyed, and it can't be inadvertently erased.

AppleCD SC features

The AppleCD SC, shown in Figure 1-1, provides a CD-ROM solution for both the Macintosh and Apple II computers.

■ **Figure 1-1** AppleCD SC features



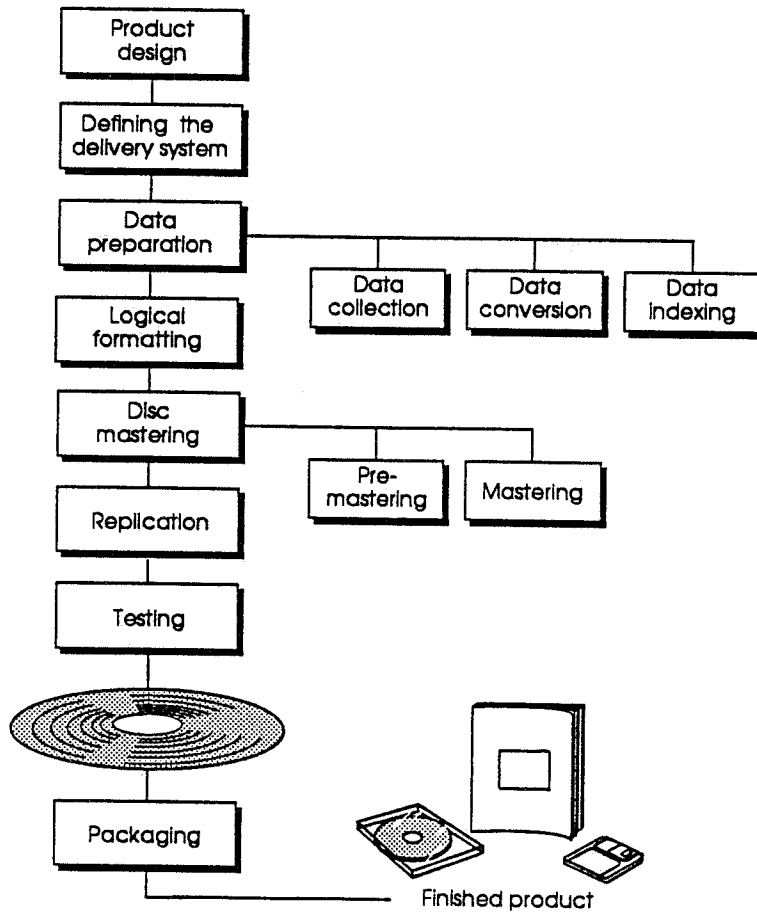
The AppleCD SC has the following features:

- The AppleCD SC has the same dimensions of width and depth as the Apple Hard Disk 20SC, the Apple Tape Backup 40SC, and other Apple high-capacity storage devices. The AppleCD SC's size and compact design allow it to fit under the Macintosh Plus and Macintosh SE. The AppleCD SC can also be easily stacked with additional Apple peripheral devices.
- The AppleCD SC is a **SCSI device**, which is easily connected to the Macintosh Plus, Macintosh SE, and Macintosh II, as well as the Apple IIe and Apple IIGS® configured with an Apple II SCSI card containing a ROM revision C or later with the part number A2B2087. The AppleCD SC can be daisy-chained within a group of up to seven SCSI devices, which may include additional AppleCD SC drives.
- The AppleCD SC reads CD-ROM and plays audio compact discs. In addition to the variety of CD-ROM applications available for your use on both the Macintosh and Apple II, you can also listen to any of your favorite audio CDs by using the CD Remote program and a set of externally powered speakers or stereo headphones. The CD Remote programs for the Macintosh and Apple II are supplied with the AppleCD SC. Because the AppleCD SC plays CD-Audio and reads data, programmers can create multimedia applications that incorporate stereo voice-over or musical accompaniment to enhance the user's experience.
- For playback of compact discs and CD-Audio tracks on a CD-ROM, the AppleCD SC has a pair of stereo jacks for connection to amplified speakers, a headphone jack, and a volume control.
- The AppleCD SC has a built-in 64K data buffer to maintain a high data-transfer rate to the host computer.
- The AppleCD SC supports the Philips/Sony CD-ROM *Yellow Book* industry standard and the CD-Audio *Red Book* standard.
- The AppleCD SC system software supports the High Sierra and ISO 9660 industry standard CD-ROM formats.

Chapter 2 **Creating a CD-ROM**

Certain conventions currently exist for CD-ROM product creation, and this chapter attempts to categorize and explain the process generically. Some steps may not be necessary in an individual case. For example, if your CD-ROM product is a set of HyperCard stacks, you may not need to perform the data indexing step (described later). The step-by-step process to create a CD-ROM is presented in Figure 2-1 and described in the text following the figure. ■

■ **Figure 2-1** The steps to create a CD-ROM



Product design

Your product design is the key point on which you base your future decisions. It also helps to ensure that your first-time results are in fact your intended results, so that you don't have to go through the process again.

A good product design focuses on a particular audience. As you define the audience your product will be aimed at, you must first thoroughly analyze the audience's potential *market share*—in other words, is the audience interested in your product large enough that you can get back your development costs and still make a profit? Consider also what other, competitive products already exist for that audience. From this analysis you can decide what information, features, and functions will be required to make your product successful with your chosen market audience.

The product design also includes considerations of disc contents, product price, packaging, distribution channels, and the estimation of time and effort required to meet your target goals. To help you with your decisions about the product design, consider these questions:

1. How is the information currently presented? Is the information in the form of books, disks, monthly periodicals, and so on?

Consider how you can improve your product from an ease-of-use standpoint. If the user is currently thumbing through large volumes of information to find a specific topic, will the design for your user interface and search mechanism be easy enough to use to make the customer want your product as an alternative to the current product?

2. If you already have a large database of information or are creating one, think about what audience or audiences see the information as valuable.

Consider what the audience or possible combination of audiences need. For example, a bicycle dealer would find exploded views of mechanical parts useful, but a list of related repair manuals, bicycle tour guides, bicycle clothing, and accessories books would make your product more valuable to the initial audience, the dealer, and in turn his audience, the customer.

3. How does the audience use the information as it is currently presented?

For example, in a library, the card catalog is large and requires that you physically thumb through cards to find what you are looking for.

4. How can the use be improved on CD-ROM?

If you put a library card catalog on a CD-ROM, the CD-ROM catalog would be physically smaller than a conventional card catalog. The user would be able to work in front of a computer with an extensive help system to guide them through the catalog. A CD-ROM could also be set up in a multi-user configuration, allowing many users to access the same cards, at the same time.

5. How can the information be economically converted into machine-readable form for use on CD-ROM?

- Does it need to be converted from tape? Electronic data, video, and audio may be on tape.
- Does the information have to be keyboarded on a word-processing system, or can the information be captured by other techniques, such as using an optical character reader (OCR) or, in the case of graphics, image scanning?
- Does the information require formatting for screen presentation and editing for style, length, correctness, and so on?

6. How often will the information need to be updated, and how will updates be handled? Will the product be a service with monthly or quarterly updates, or will you notify the customers of an update, giving them the option to purchase the new material?

A monthly service is generally more expensive for customers, because they pay for the extended update service up front. The expense may limit your product to larger corporate or government accounts. The second alternative of providing optional updates opens your market to local small businesses and city agencies.

7. Will you have to license the information from an information provider?

Licensing information can sometimes save you considerable time when compared to compiling information from multiple sources. It also gives you more freedom to concentrate on the user interface for your product.

8. What kind of retrieval software will best serve your customer? (**Retrieval software** is the software that does the actual search for the information that the user requests from the CD-ROM.) Will you create customized retrieval software or license a general-purpose package from a software company that specializes in retrieval software?

9. Will the retrieval and application software be provided on the CD-ROM or on floppy disk?

It is generally best to have the software distributed on floppy disk, because you may want to have the flexibility to upgrade without mastering another series of CD-ROM discs.

10. Will you license the product for individual sites or sell it outright to the customer to package and distribute?
11. How will the product be packaged? What resources do you have for producing special graphics for labels and packaging?

If you have no experience in this area, the time and expense of bringing graphics personnel and equipment in-house for smaller projects would be better spent on data organization and your user interface design. This phase of the project is probably best left to a professional marketing and packaging company. You would need only to convey your ideas to the professional service, then review and approve their conceptualization, rather than spend time and money working on an unfamiliar subject.

12. How will you distribute the product: through computer dealerships, software outlets, or direct sales? Will you instead use it exclusively within your own company?

Another key element for a successful product designed to work in the Apple computer market is the user interface. The importance of the interface really can't be over-emphasized. Apple computer users expect an interface that

- extends the Apple desktop metaphor (described in the *Human Interface Guidelines: The Apple Desktop Interface*, which is available from Addison-Wesley)
- is rich in graphic elements
- provides ease of integration into the user's work environment by being consistent and following the Apple interface guidelines
- is intuitive and requires very little tutorial documentation, so that the occasional user and professional user both feel equally comfortable working with the product
- incorporates compatibility for future updates

Defining the delivery system

During this stage of the CD-ROM creation process, you analyze how your CD-ROM information is going to be delivered to the user. Here the term *delivery* refers to hardware and software delivery, rather than distribution and sales delivery. The target delivery system for CD-ROM products for the AppleCD SC is either a Macintosh computer or an Apple II computer. Typically, you must focus on these questions during this stage:

1. What type of software will make the product valuable and exciting to the customer?
2. Who will design and maintain the software?
3. What is the target machine? If you are an information provider, you may want your information to be accessible by more than one brand of computer.
4. Does your product require the use of special hardware, such as a large-screen color monitor or a laser printer? Will your target audience be willing to purchase a product that requires special equipment?
5. Will you provide a complete CD-ROM solution, which includes the computer hardware delivery system and your CD-ROM?

If you plan to market a complete CD-ROM solution, which includes system hardware and software, here are some additional marketing issues that will require your attention:

1. How will you deliver the hardware? Will you sell it outright, lease it, or package it with a product subscription?
2. Where will you procure the system components? Will you purchase the components from a system integrator or buy them directly from a dealership or manufacturer?
3. How will the system be serviced? What warranty will be available?

Software development

Software development for CD-ROM encompasses several areas. You need to decide what areas of the software you intend to develop yourself and what areas should be contracted or licensed from other sources. Having an in-house programmer (or programmers) who understands the target computer and operating system is ideal for software product support and fast turnaround when developing fresh ideas. If you are a software developer, particularly a developer who has not worked with Apple computers and software before, you also need to decide whether to bring Apple expertise in-house for your project.

Regardless of whether you bring the software expertise in-house or contract on the outside, you still need to analyze how the retrieval software will be handled. Will you start from scratch and create your own retrieval software, or license general purpose software and enhance the functionality for your application?

The amount of work for each area of software development will depend largely on the specifications of your product design. The list that follows provides a good preliminary outline of the areas of software development where you need to concentrate your efforts.

- Development of the user interface for either the Macintosh or Apple II families of computers or both. (*The Human Interface Guidelines: The Apple Desktop Interface*, which is available from Addison-Wesley, provides an excellent reference for implementation of the Apple desktop interface.) An example of a desktop interface that follows the ideas presented in the *Human Interface Guidelines* reference is provided in the section "A Friendly Interface," Chapter 4.
- Programming the retrieval software (sometimes referred to as the *retrieval engine*) for a new Apple product, porting an existing retrieval engine so that it works on either the Macintosh or Apple II, or licensing existing retrieval software that has been created for the Macintosh or Apple II. A description of CD-ROM retrieval engines is provided in the section "HyperCard as a Data-Retrieval Engine," Chapter 4.
- Programming device resources (commonly referred to as *device drivers*) for application output. If your CD-ROM product design calls for the application to send output to and control devices not currently supported by Apple device resources, you need to provide device-specific resources for your application. For example, output devices might include typesetting equipment, printers, plotters, and so on.
- Building the index for your CD-ROM retrieval software. The index is a key element for locating words on the CD-ROM during the search process executed by the retrieval software.

- Localizing your interface for foreign countries. This would include developing or purchasing special character sets and translating the language in your database and interface. Information for localizing Macintosh applications can be found in *Inside Macintosh*, and the *Human Interface Guidelines*, published by Addison-Wesley, and *Software Development for International Markets*, available from APDA.

The interface, the retrieval engine, the information, and the index are the pieces of your software development effort that together make up your application. To produce an efficient CD-ROM application, these phases of software development must be carefully thought out and closely tied to each other.

Data preparation

The data preparation phase includes several steps, some of which require considerable expertise. As you saw in Figure 2-1, the data preparation step in the CD-ROM creation process is generally divided into three areas:

- data collection
- data conversion
- data indexing

Data collection

This step is often referred to as the *data capture step*, because it includes both the creation and the transformation of original data from diverse sources, and conversion of that data into one format. For example, the data could be binary files, line drawings, photographs, film, or word-processing files.

To get the data into a usable form, you first collect and convert all dissimilar types of data into a format that can be read by a computer. This conversion may involve digitization of analog sound, graphics, text, videotape, and so on.

The data capture step is one of the areas that may require expertise from outside vendors. Getting data, which may exist only in books, into machine-readable form requires many hours of keyboarding the data into a computer. If you are not set up to do a lot of keyboarding, this task is better left to vendors that specialize in word processing. If resources and the proper equipment are available, conversion of photocomposition tapes, digitization of analog sound, OCR of text data, and scanning of graphic images can be done in-house.

Data conversion

After the data has been converted into a machine-readable form, it may still not be in a form that the retrieval and indexing software will be able to use. During the data conversion stage, you reformat and code the data to be read, displayed, and indexed. For example, words, paragraphs, pages, chapters, and section markers may have to be encoded so they can be indexed for your retrieval software. Additionally, you should correct any errors in the data. Typical errors you want to catch and correct are all spelling, formatting, style, and outdated-material errors. Error correction during the data conversion stage is important, because the next phase of data preparation, data indexing, will index all overlooked errors along with the good data. Catching and correcting the errors later in the production cycle will be costly and cause delays in your project schedule.

You will also need to monitor the total size of your text, graphics, and sound data. If the quantity of data exceeds the size of your disc, you may need to consider data compression. Remember that if you do compress the data, it will have to be decompressed later when it is displayed on the screen. The data compression stage is done with software and in some cases with hardware. Custom compression and decompression software and hardware may add considerably to your production costs and the cost of the product to the user. Data decompression also adds to the data retrieval time.

If you plan a product that includes sensitive data or programming data that you want to protect from being easily copied, you will also want to encrypt the data during the conversion process. Here again, the data encryption process requires the use of special encryption and decryption software, which may add to the cost of the final product. Data decryption also adds to the data retrieval time.

Data indexing

In the same way that an index for a book allows you to find information quickly, data indexing allows the retrieval software to quickly locate information on a CD-ROM. The data index provides a table of word occurrences and the location of each word on the CD-ROM. (Graphics, such as figures within text, can also be indexed.) Because of the quantity of information and slower access times of the typical CD-ROM, it is necessary that all data be indexed; otherwise, information retrieval could take an unacceptable amount of time.

The structure of the index is closely tied to the retrieval and application software. Indexing schemes vary according to the type of information and presentation being provided to the user. For example, a full-text reference product that presents formatted pages of information to the user would require a list of every word in every chapter head, paragraph, and sentence. Each application design may require a slightly different method of indexing to make it efficient and useful with the information. (A simple example for building an index with information in a Macintosh HyperCard stack is provided in Chapter 4, "HyperCard and CD-ROM.")

Logical formatting

This stage comprises both the logical arrangement and logical formatting of data. During the first step, you need to test and optimize your retrieval software and to arrange logically the layout of data on your hard disk or disks. Data often needs to be rearranged or physically placed on the disk to improve application performance. Many CD-ROM development subsystems allow you to perform real-time simulation of the CD-ROM environment on disk and to build a logically formatted tape.

The logical formatting step places all of the text, graphics, processor sound, or other forms of data into a unified file structure onto a writable medium, often a 9-track tape. Your choices here are to use a format readable by the target machine, for example, the ProDOS file format for the Apple II family, or the HFS file format for the Macintosh family, or a format that is accepted universally. If you elect to use either of the Apple formats, your disc will become machine dependent. Alternatively, you could convert all the data into the ISO 9660 format, which would make the data files on the disc readable by most of the computers that support the CD-ROM ISO 9660 format. (The High Sierra and ISO 9660 formats are discussed in Chapter 3, "High Sierra/ISO 9660 and the AppleCD SC.")

The logically formatted tape should be an ANSI-compatible tape or tape set formatted to Level 3 of the ANSI *Standard for Magnetic Tape Labels and File Structure for Information Interchange* (X3.27-1978). The information on the tape should be an exact data image of what the CD-ROM manufacturer will transfer to the CD-ROM master tape. The information from the tape will be combined with any CD-Audio during the disc mastering stage.

You should also be gathering any CD-Audio content that you plan to include on the CD-ROM. The sound studio you are working with records your sound segments on tape in a CD-Audio format. You need to supply the CD-ROM mastering facility with both your CD-Audio tapes and your data tapes for creation of the CD-ROM master tape. CD-ROM mastering facilities can supply more information about the requirements for both data and CD-Audio tape formats. (For more information about CD-Audio sound, see Chapter 5, "AppleCD SC and Sound.")

Disc mastering

The CD-ROM disc mastering process includes these steps:

- tape premastering
- mastering the disc

Tape premastering

If resources are available, you can use one of the microcomputer-based premastering subsystems for CD-ROM development. Micro-based subsystems allow you to prepare the premastered tapes in-house.

If the appropriate software is available, premastering can be performed on any number of computer systems that have large disk and tape storage capabilities. Large-capacity tape drives, which can be used for premastering, are available for the Macintosh computer.

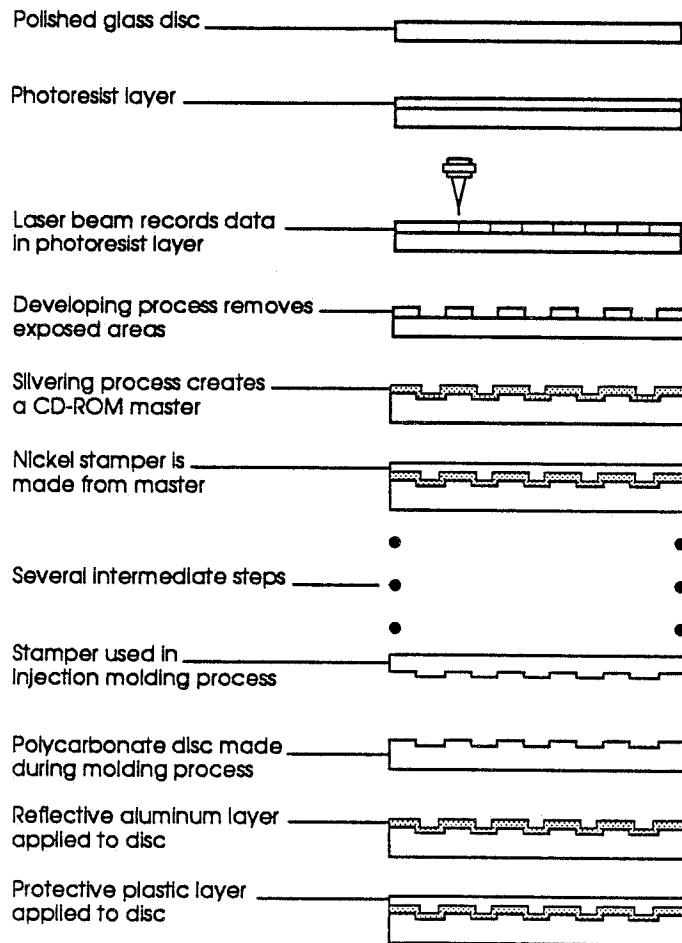
After your premastered tape has been made, you send it to a CD-ROM mastering and manufacturing facility (also referred to as a CD-ROM publisher). The mastering facility processes the data on the tape, adding error detection and correction information. The combination of data and error correction information makes up the master tape for the final CD-ROM format, which will be used in the CD-ROM mastering process. If you are including CD-Audio on the disc, it will also be combined on the master tape.

Some CD-ROM publishers also accept information stored on hard disk drives as an alternative to premastered tapes. You need to ship the hard disk with all of your content for the CD-ROM to the publisher. The CD-ROM created from this process is an exact data image copy of the hard disk, so be sure that all of the data is organized exactly as you want it on the CD-ROM before you ship the hard disk. Contact several CD-ROM publishers to find out if they provide this service.

Mastering the disc

A simplified diagram of the CD-ROM mastering process is shown in Figure 2-2. The mastering process starts with a glass disc that is coated with a photoresist layer. Using extremely fine-tolerance equipment, a laser beam is modulated synchronously with the digital information on the premastered tape. The modulated laser beam exposes the photoresist layer on the surface of the spinning glass disc and records the digital information. Photodeveloping chemicals are applied to the exposed photoresist layer, which cleans the glass master, leaving only the digital imprint created by the laser beam. Then a nickel layer is applied to the disc, creating a metal master stamper. To preserve the master stamper, several additional metal stampers are made for later use in CD-ROM replication.

■ **Figure 2-2** The steps involved in the CD-ROM mastering process



Replication

During disc replication, the metal stampers are used in a transfer-molding process to make an exact imprint of the information surface on polycarbonate discs. The polycarbonate discs are coated with a thin reflective aluminum layer, as shown in Figure 2-2. The aluminum layer is then coated with a protective plastic, and the CD-ROM label is silk-screened onto the plastic layer side of the disc.

If you are making your first replication run, you may only need a minimum number of CD-ROM discs for testing. After the testing process has been completed and any errors corrected on your master, you can then start full-scale replication of the discs.

Testing

At this stage you perform all of your testing with the actual CD-ROM. The CD-ROM testing stage is your last chance to iron out any potential problems with the interaction between your retrieval application and data. Many things may not interact exactly as they did before everything was placed on the CD-ROM. You may have to rearrange your data or enhance some parts of your application to optimize product performance. Any modifications you make to your product must be run through the mastering and replication stages again.

Never underestimate the amount of time needed for testing your CD-ROM. Include plenty of time in the product design for testing.

Packaging

This stage puts the CD-ROM product together in the finished form with all of its pieces. The packaging pieces generally include the package the CD-ROM will be contained in, the user documentation for the product, warranty and additional services documentation, and the shipping container. Many of the CD-ROM mastering facilities provide a service that takes all the pieces of your CD-ROM product, puts the pieces together, and ships the finished product to your distribution center or end user.

Equipment and software for product development

This section includes information for software developers getting started on developing software for the Macintosh and Apple II families of computers.

Developing software for a Macintosh

If you plan to develop CD-ROM products for the Macintosh Plus, Macintosh SE, or Macintosh II, the combination of hardware and software that provide the fastest turnaround for the smallest investment is a Macintosh II configured with a minimum of 2 megabytes of memory, a monochrome monitor, an extended keyboard, an Apple Hard Disk 80SC, and whichever Macintosh target machine you plan to run the software on. If your product is targeted for the Macintosh II and you intend to use color graphics as part of your product, you need to add the Macintosh II color graphics card configured with extended memory and an Apple High-Resolution RGB Monitor or other color monitor compatible with your system.

The ideal development software for the Macintosh is the Macintosh Programmer's Workshop (MPW™) Shell. The MPW Shell is a robust, integrated, window-based environment for program editing, file manipulation, linking, and program execution. The system also includes many other tools that create and manipulate text and **resource** files. C, Pascal, and assembly languages are available for the MPW Shell. Brief descriptions of the MPW tools and languages are as follows:

- Macintosh Programmer's Workshop C: A C compiler that runs in the MPW Shell. It provides the tools, interfaces, and libraries to develop applications written in C.
- Macintosh Programmer's Workshop Pascal: Provides the tools, interfaces, and libraries to develop applications written in Pascal.
- MacApp®: An expandable generic application that can greatly simplify and speed the process of software development. MacApp consists of a set of object-oriented libraries that implement the standard Macintosh user interface in such a way that you can write applications as extensions to MacApp.

Depending on the programming language you'll be using, you may want to order one or more of the reference manuals listed below.

- *Inside Macintosh*, Volumes I, II, IV, and V
- *MPW Reference*
- *MPW Assembler Reference*
- *MacApp Programmer's Reference*
- *MPW Pascal Reference*
- *MPW C Reference*

MPW, the accompanying tools and languages, and documentation are available from APDA.

Development time for the user interface portion of your Macintosh CD-ROM product can be greatly reduced with the use of HyperCard. To learn more about how HyperCard can help you to quickly build Macintosh user interfaces, see Chapter 4, "HyperCard and CD-ROM."

Developing software for the Apple II family

If you plan to develop CD-ROM products for the Apple II, the hardware that provides the fastest turnaround for the smallest investment is an Apple IIGS configured with 1 megabyte of memory, a monochrome monitor, a keyboard, an Apple Hard Disk 80SC, and the Apple II target machine. (Because the maximum size of a ProDOS volume is 32 MB, you will have to partition larger hard drives into a number of smaller volumes.)

The ideal development environment for the Apple IIGS is the Apple IIGS Programmer's Workshop (APW™), available from APDA. APW is a complete software development system. It includes a command shell, linker, and utilities. It is also a host for many APW language products. The command shell provides file management, directory listing, I/O redirection, and pipelining, as well as utilities with useful extensions to ProDOS 16 and interfaces providing call macros and equates for the GS/OS™. A full-screen text editor features copy, move, search and replace, and deletion of blocks, and executes editor command macros. APW requires 1.25 MB of RAM and two 3.5-inch disk drives or one 3.5-inch disk drive and a hard disk. Brief descriptions of the APW tools and languages are as follows:

- **APW C:** Apple IIGS Programmer's Workshop C v.1.0.2 generates APW object files and includes extensions for void types, enumerated types, and structure passing. It supports source-level segmentation of load files and includes standard C I/O library and Apple IIGS tool interfaces.
- **APW Interface Update:** This package allows users to update older versions of APW to the version 1.0.2 assembler interfaces.

You can also develop Apple IIGS software in the Macintosh MPW environment with the Macintosh to Apple IIGS Cross-Development System. A full-featured macro assembler, MPW IIGS Assembler, and C compiler, MPW IIGS C v.1.0.1, are available from APDA.

Chapter 3 **High Sierra/ISO 9660 and the AppleCD SC**

This chapter includes a brief overview of the High Sierra and ISO 9660 file and volume formats, and describes how the Macintosh or Apple IIGS presents High Sierra and ISO 9660 files and directories to the user. It also provides a comparison of the High Sierra/ISO 9660 and Macintosh HFS. ■

The High Sierra and ISO 9660 file systems

High Sierra and ISO 9660 are standards that specify a hierarchical volume and file structure for CD-ROM discs. The purpose of these standards is to provide a CD-ROM file structure that is not dependent on a particular operating system. The High Sierra and ISO 9660 file systems allow for the interchange of information between any number of computer operating systems that are configured to recognize and translate the High Sierra and ISO 9660 file and volume structures.

The High Sierra standard came about when a group of industry representatives met at Del Webb's High Sierra Hotel and Casino in Stateline, Nevada, in late 1985 to cooperatively develop a common logical format for CD-ROM discs. The result of this series of meetings was a standard known as the "High Sierra" standard. This standard is fully specified by the May 28, 1986, *Working Paper for Information Processing—Volume and File Structure of Compact Read Only Optical Discs for Information Interchange*. For obvious reasons, this is known as the High Sierra paper.

As is the case with all good standards, the world at large wanted to adopt an equivalent standard. The International Standards Organization (ISO) has modified the High Sierra standard by running it through the ISO standardization process. The end result is a new international standard, *ISO 9660—Volume and File Structure of CDROM for Information Interchange*. This is known as the ISO 9660 standard. Although most existing CD-ROM discs are High Sierra, ISO is the wave of the future, and most future discs will be ISO. Throughout this section, *ISO* refers to both ISO 9660 and High Sierra unless explicitly stated differently.

ISO 9660 and High Sierra differ only in minor details. Some fields in the volume descriptor have been changed. A couple of extra occurrences of the optional path table were deleted. A bibliographic preparer field was added. The date and time structures have a new field, which includes a 15-minute offset from Greenwich Mean Time, for true internationalization of time values.

CD-ROM discs all use the same Philips/Sony format for the physical organization of the disc (as specified in the Philips/Sony *Yellow Book*). This means that no matter who creates the disc, every CD-ROM disc drive can read the raw bits. You can contrast the CD-ROM format to the current format of the Apple 3.5-inch 800K disk, which uses a different encoding scheme from that of any other manufacturer; therefore, most non-Apple floppy disk drives can't even read the bits on an Apple 800K disk. Having a physical format in common is nice, but it's not enough. A common logical format is required for a variety of machines to be able to read and retrieve specific files on a disc. ISO 9660 is a common logical format for CD-ROM discs.

ISO 9660 design goals

ISO 9660 has these basic design goals:

- Provide easy implementation under a variety of operating systems, such as Macintosh, various Apple II ProDOS, MS-DOS, A/UX®, VMS™, and so on.
- Optimize the design for read-only media.
- Account for the relatively low seek time of CD-ROM drives.

ISO 9660 does not define the contents of a file. File content is an application issue.

The Philips/Sony *Yellow Book* standard defines how data is physically stored on a CD-ROM disc. The physical sector contains 2336 bytes, of which 288 bytes are used for error correction. The remaining 2048 bytes, or 2K, are user data. The ISO standard uses this 2K sector as its logical sector.

Logical sectors can be divided up into one or more logical blocks. A logical block must be a multiple of 512 bytes. Most CD-ROM discs are created with the logical block size equal to the logical sector size.

ISO 9660 file-naming conventions

ISO 9660 provides for naming files in one of two ways:

- With a subset of ISO 646 characters, which is essentially 7-bit ASCII. Valid characters are the uppercase letters A through Z, the digits 0 through 9, and the underscore (_).
- With any coded character set. This allows for a language with a non-Roman alphabet, such as Japanese and Arabic. You can use any character set you want, as long as it meets the needs of your end users.

Using a coded character set does not guarantee that the CD-ROM is usable by other applications, so most publishers restrict filenames to the ISO 646 subset.

File identifiers consist of a filename, followed by a filename extension, and optionally followed by a file version number. The filename and filename extension are separated by a period. The filename extension and file version number are separated by a semicolon. Either the filename or the filename extension must exist. The file version number is optional.

Filenames can conform to any of three levels of file under High Sierra. Level 1 conformance allows files to have names conforming to the MS-DOS standard: an 8-character filename plus a 3-character filename extension. The other levels of conformance allow 31-character file naming, including the total of the filename, filename extension, and file version number. ISO has no such conformance levels; file identifiers can be 31 characters long. In practice today, most publishers restrict filenames to the 8-plus-3 convention of MS-DOS for easy interchange of information.

Directories are special files, with their content specified: collections of directory records. A *directory record* defines the file identifier, size, starting logical block number, and other information used to access the file. ISO 9660 supports subdirectories, and you can nest subdirectories up to eight levels deep.

Scanning the directory tree to find a specific subdirectory can take a long time on a CD-ROM, because of the low seek times of CD-ROM drives. To help speed up the search process for CD-ROM, ISO 9660 defines a path table, which is an index providing the location of all the directories.

ISO 9660 also defines *associated files* (used, for instance, in the Macintosh world to store the resource fork of a file) and *hidden files* (used in the Macintosh world for invisible files).

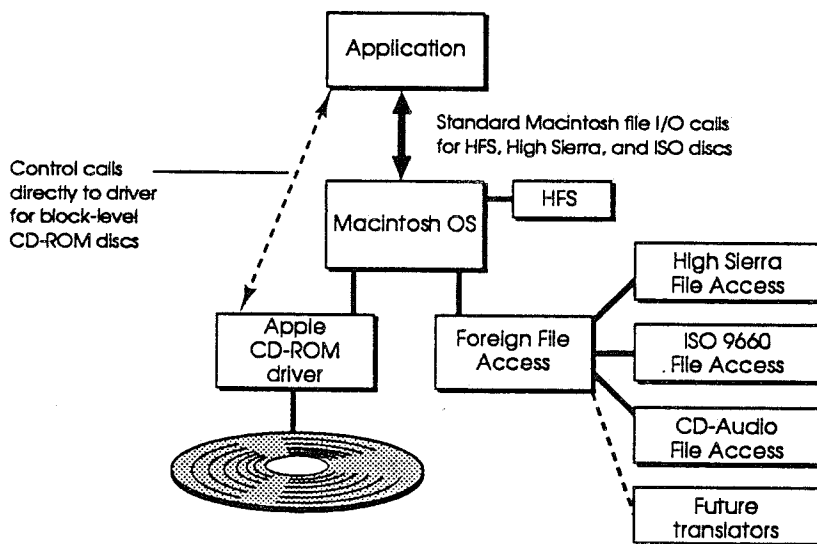
For further reading about ISO or High Sierra, see the two standards papers mentioned earlier in this document. To obtain these papers, contact the NISO, National Bureau of Standards, Administration 101, Library E-106, Gaithersburg, MD 20899. Additionally, for a good overview of the High Sierra file system structure, see *CDROM 2: Optical Publishing*, available from Microsoft Press.

ISO 9660 on the Macintosh

The Macintosh Foreign File Access software, in conjunction with the specific file system access software, provides generic support for recognizing file system formats other than the hierarchical file system (HFS). Foreign File Access passes disc identification information received from the Macintosh operating system to the file system access software modules to test whether they can read the disc. The Audio CD Access file provides support for putting an audio CD icon on the desktop to preserve the Macintosh interface. Other file system formats may be supported in the future.

The block diagram in Figure 3-1 illustrates the software modules that interact to read a CD-ROM. A description of the interaction follows.

■ **Figure 3-1** The CD-ROM software environment



The following sequence of events takes place on the Macintosh when a CD-ROM is inserted in the AppleCD SC.

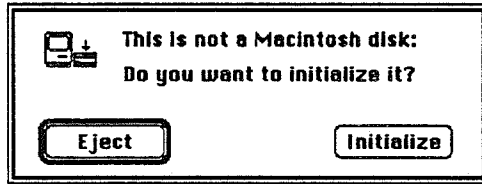
The AppleCD SC device driver recognizes that a disc has been inserted and notifies the Macintosh Operating System (OS) that a disc is in the drive. The Macintosh OS calls HFS. HFS tries to read the disc in the drive and to identify it as an HFS-format disc. If the disc is in HFS format, all file I/O is directed through HFS. If the disc is not HFS, the Macintosh OS looks for Foreign File Access software. If Foreign File Access software exists, it is called by the Macintosh OS.

When the Foreign File Access software gets control, it passes disc identification information to the file system access software, to High Sierra/ISO 9660 File Access, or to Audio CD Access, waiting for acknowledgment that one of them recognizes the disc. The file system access software returns an error if it cannot recognize the disc. If one of the file system access files does recognize the disc, it returns `NOERR`. Once a file system translator acknowledges the disc, all file I/O for the disc is handled through the Foreign File Access software and the specific file system access software for the file system on the disc.

- ◆ *Note:* Your application sees the CD-ROM disc as any other disk connected to the system. All file I/O calls are standard Macintosh calls. Nothing special needs to be done. For example, if you want to read a file from a High Sierra or ISO CD-ROM, issue a `Read` command.

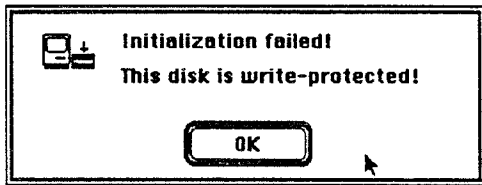
If no file system access software recognizes the disc, no other foreign file system software recognizes the disc, and the Macintosh OS can't recognize the disc, you get an Eject/Initialize dialog box similar to the one shown in Figure 3-2.

■ **Figure 3-2** An Eject/Initialize dialog box



Because CD-ROM is read-only media, you won't be able to initialize it. The Macintosh returns an Initialization failed dialog box, as shown in Figure 3-3, leaving the OK option as your only choice. If you click OK, the CD-ROM is ejected and you return to the previous process.

■ **Figure 3-3** Initialization failed dialog box

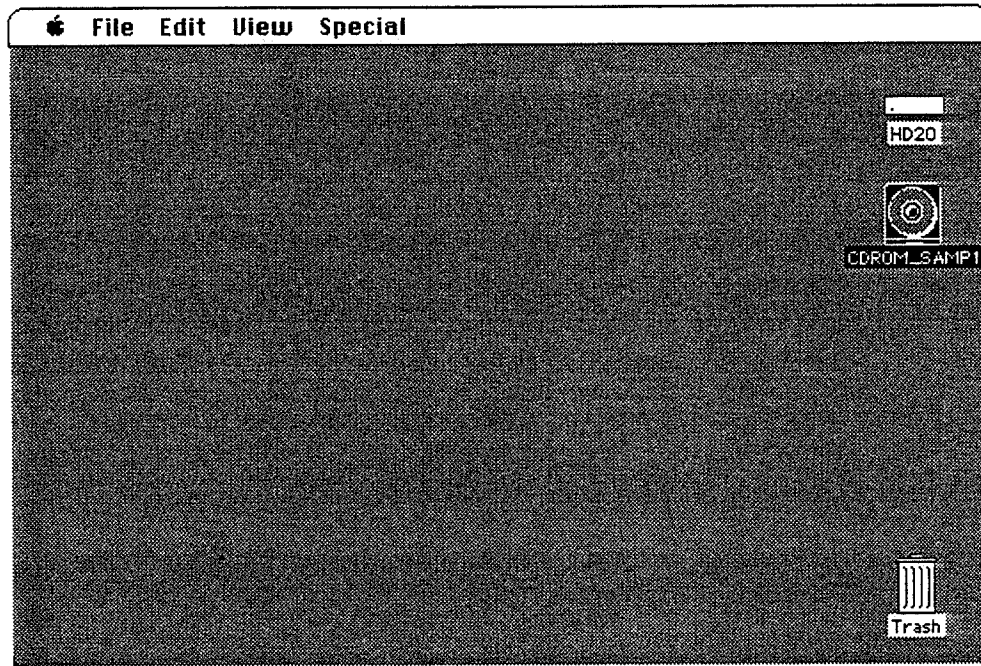


The Macintosh OS doesn't handle CD-ROM discs with unsupported file systems, such as block-level CD-ROM discs. If you intend to create block-level CD-ROM discs, your application will have to handle the disk-insertion events, initialize the disk environment, and make all device control and status calls through the Apple CD-ROM driver. (For information about how to get the disk-insertion events and how to handle such events in your application, see the Event Manager chapter and disk initialization package section in *Inside Macintosh*, Volumes I, II, and IV.)

High Sierra and ISO 9660 discs on the Macintosh desktop

If you have installed the Foreign File Access system software and have an AppleCD SC connected to your Macintosh, place a High Sierra or ISO 9660 CD-ROM in the caddy and insert the caddy in the AppleCD SC. The CD-ROM icon will appear on the desktop, as shown in Figure 3-4.

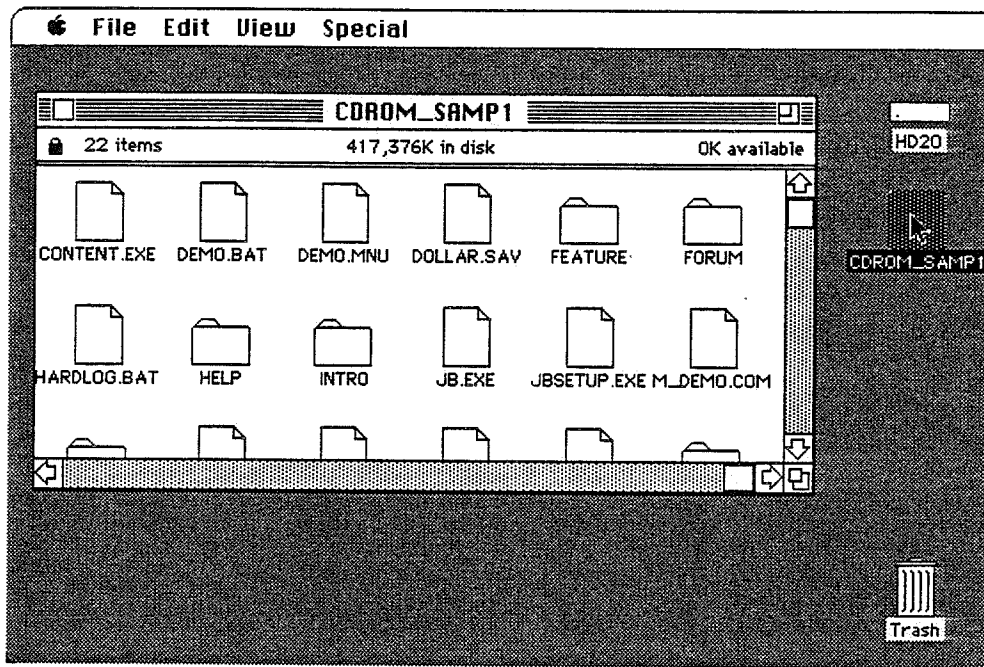
■ **Figure 3-4** The CD-ROM icon on the desktop



- ◆ *Note:* The amount of time it actually takes for the CD-ROM icon to appear on the desktop is proportional to the amount of files on the High Sierra/ISO 9660 CD-ROM.

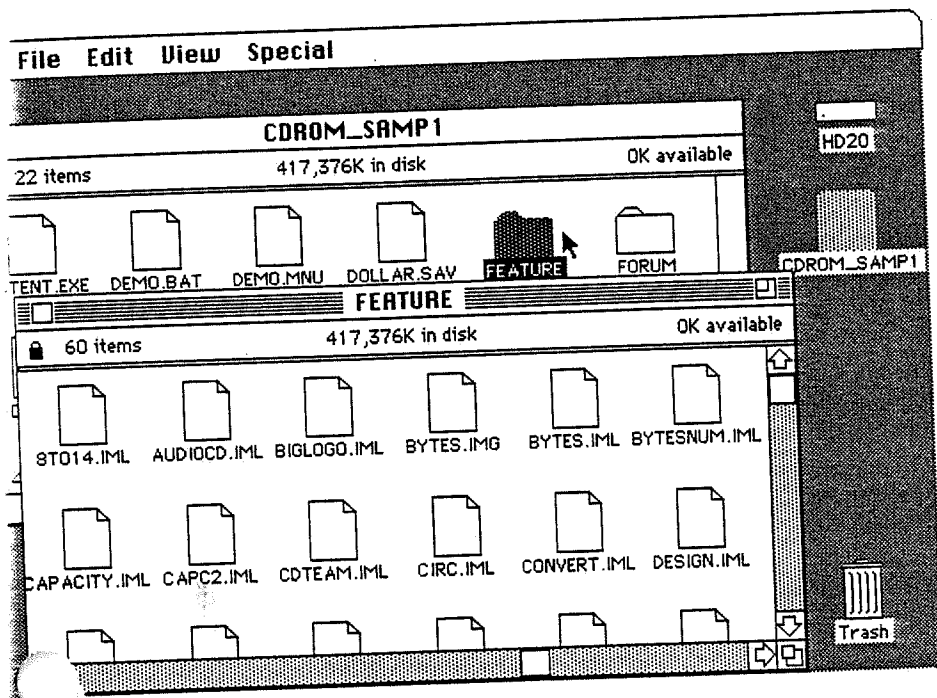
The Foreign File Access software, in connection with the High Sierra File Access and ISO 9660 File Access files, allows you to mount and open High Sierra and ISO 9660 formatted CD-ROM discs. The contents of those discs can be viewed just as any other Macintosh 3.5-inch disk or hard disk formatted with the HFS file system. Figure 3-5 shows the contents of a High Sierra CD-ROM viewed in the By Icon mode, selected from the View menu. The other View menu modes are also supported.

■ **Figure 3-5** Viewing the contents of a High Sierra disc



All files (text or application) appear with the default text file icons. All files on a High Sierra or ISO 9660 disc are of type 'TEXT', creator 'cdhs'. All directories appear with the default file folder icon. You can use a text editor to open files that you know contain ASCII text. You can open and view the contents of directories and subdirectories, as shown in Figure 3-6.

Figure 3-6 Viewing the contents of a directory on a High Sierra disc



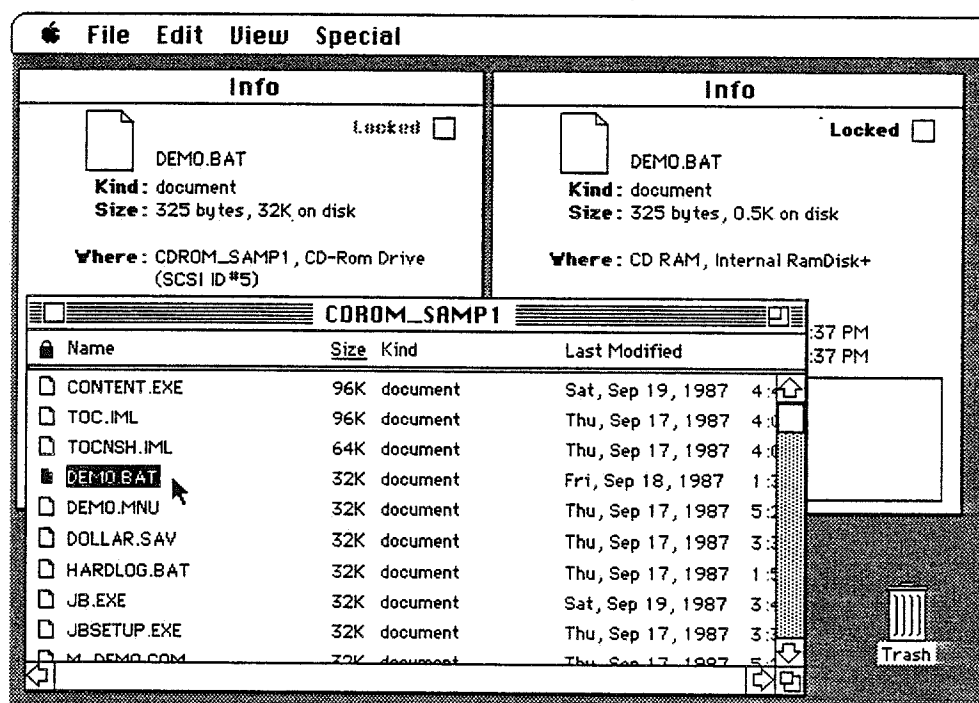
Because the CD-ROM disc is a locked medium, the remaining available space on the disc always shows 0. Additionally, the CD-ROM icon appears on the desktop without any of its windows open when a new disk has been inserted or when you quit an application and return to the Finder™. (If you are using MultiFinder™, this may not be true. Other applications could be open on the CD-ROM.)

Audio CDs are also recognized by the Foreign File Access software. When you insert an audio CD in the AppleCD SC, an icon appears on the desktop, indicating that a disc has been inserted. You can eject an audio CD by dragging the Audio CD icon to the Trash, or by pressing Command-E. If you double-click the Audio CD icon, a window is displayed showing file icons for each audio CD track on the CD.

Additional information about ISO, Macintosh, and the AppleCD SC

- The Foreign File Access software does not support unique icons or files or applications that can be opened from the Macintosh desktop for Macintosh HFS files existing under the High Sierra or ISO 9660 format. The Finder depends on information that exists in the Desktop resource file to find the appropriate resources for displaying icons and locating applications to be launched from their associated documents. Premastering software to build the Macintosh Desktop file to be placed on a High Sierra or ISO 9660 disc has not been created.
- On any large volume, such as CD-ROM, the value reported for a file's size on disk is artificially high in the Get Info dialog box and in the View By Size volume window. However, the total bytes value is correct. An example is shown in Figure 3-7. The Get Info dialog box on the right contains the size values reported for the Demo.bat file after it was copied to another disk. Note the value for the size on disk; it corresponds to the 325-byte value. In the Get Info dialog box on the left, the value of 32K on disk is artificially high. The same is true in the size field of the CD-ROM volume window, CDROM_SAMP1, when viewed by size.

■ **Figure 3-7** File size shown in the Get Info dialog box and CD-ROM volume window



- AppleShare® 2.0 supports locked media, such as CD-ROM, on the AppleTalk® and EtherTalk™ local networks.

High Sierra and ISO 9660 CD-ROM discs on the Apple IIGS

The High Sierra file system translation software for Apple IIGS systems supports single High Sierra volumes on a CD-ROM. The High Sierra volumes can be of any size, as long as only one volume exists on the disc.

Files and directories on ISO discs will appear on the desktop with the standard default file and folder icons. You will be able to display files and folders in each of the modes available from the View menu. More information about the application interface and implementation-specific details are included in the Apple *GS/OS Reference* and are available from Apple Developer Technical Support.

High Sierra and ISO 9660 CD-ROM discs on the Apple IIe

A set of High Sierra/ISO 9660 utility routines exist for the Apple IIe computer. These routines allow applications to read High Sierra/ISO 9660 CD-ROM discs. The Apple IIe High Sierra/ISO utility routines must be specifically called by the application. For implementation-specific details, contact Apple Developer Technical Support.

Comparing the High Sierra and ISO file systems with HFS

This section includes a brief comparative description of the native Macintosh HFS and High Sierra and ISO 9660 file systems. The differences pointed out in this section may help you decide what format you want to use when placing your information onto CD-ROM.

Three methods of logical data organization, or formats, are commonly used on CD-ROM discs that are created for the Macintosh environment. These are: block or absolute level, native Macintosh HFS, and High Sierra/ISO 9660. Each of these formats has specific advantages and disadvantages when used in the Macintosh environment; Table 3-1 lists some of them for comparison.

■ **Table 3-1** CD-ROM data organization comparison

Description of feature	Native HFS	Block level	High Sierra/ISO 9660
User interface	Very good	Poor	Good to very good
Data portability	Poor	Good	Good
Application performance	Modest	Very good	Very good
Ease of creation	Easiest: most software support	Easy with software	Easy with software
AppleShare compatibility	Yes	No	Yes

Using the table as a guide, you can make the following observations.

- Working with the Macintosh native HFS provides the easiest environment for CD-ROM creation. HFS has a substantial collection of software tools for application development and building the Macintosh user interface.
- The High Sierra and ISO 9660 formats provide ease of data portability across multiple computer environments, and they are structured to give the best file access performance on CD-ROM.
- Block level CD-ROM discs have no real advantages when used with the Macintosh, and they cannot be networked.

Apple extensions for ISO 9660

The information in this section is useful for

- Apple developers working with ISO 9660
- publishers of authoring tools for ISO 9660 discs
- publishers of ISO 9660 discs
- publishers of ISO 9660 receiving system software

Introduction

It may be desirable to create an ISO 9660 CD-ROM containing HFS and/or ProDOS files in order to benefit by the storage capacity and distribution cost savings of CD-ROM, and the interchange advantages of ISO 9660. However, both the HFS and ProDOS file systems require information that the ISO 9660 file system does not support: ProDOS requires a file type and an auxiliary file type, whereas HFS requires a file type, a file creator, and, frequently, an icon resource.

This document defines a protocol that extends the ISO 9660 specification to include HFS- and ProDOS-specific information, without corrupting the ISO 9660 structures. Discs created by using the protocol are valid ISO 9660 discs, and should not behave differently on non-Apple receiving systems.

In addition to preserving file-specific information, the protocol defines a mechanism for preserving filenames across the ProDOS-ISO 9660-ProDOS translation.

The protocol was designed to solve existing compatibility problems as well as to allow for future expansion. It uses the `systemIdentifier` field in the Primary Volume Descriptor for global information, and the `systemUse` field in the directory record for file-specific information.

The identifier is considered valid if the first 28 bytes match.

The Directory Record SystemUse field

Directory records in the ISO 9660 specification have the following format:

byte	DirectoryRcdLength
byte	XARlength
struct	ExtentLocation
struct	DataLength
struct	RecordingDateTime
byte	FileFlags
byte	FileUnitSize
byte	InterleaveGapSize
long	VolumeSequenceNum
byte	FileNameLength
char	FileName[FileNameLength]
byte	RecordPad
char	SystemUse[SystemUseLength]

The `RecordPad` field is present only if needed to make `DirectoryRcdLength` an even number. If present, the `RecordPad` field must be zero (\$00). The `SystemUse` field is an optional field under the ISO 9660 specification and is defined for Apple use as follows.

The `SystemUse` field, when present, must begin with a signature word, followed by a one-byte `SystemUseID`, followed by file specific information. The signature word allows a receiving system to ensure that it can interpret the subsequent data correctly, and the `SystemUseID` determines the type and format of the information that follows.

The `AppleSignature` is defined as BA (\$42 41).

Receiving systems must perform a simple calculation to determine if the `SystemUse` field is present in any given directory record. It is present if:

$$\text{DirectoryRcdLength} - \text{FileNameLength} > 34$$

Receiving systems should first verify that the `SystemUse` field is present (making sure to account for the possibility of a `RecordPad` field), then check for the `AppleSignature` before interpreting the `SystemUseID`.

The `SystemUseID` is defined as shown in Table 3-2.

■ **Table 3-2** SystemUseID

SystemUseID	Definition
\$00	reserved
\$01	ProDOS file_type and aux_type follow
\$02	HFS fileType and fileCreator follow
\$03	HFS fileType, fileCreator, bundle bit set
\$04	HFS fileType, fileCreator, and ICN# resource (128-byte icon) follow
\$05	HFS fileType, fileCreator, ICN# resource follow, bundle bit set
\$06	HFS fileType, fileCreator, finder flags
\$07–\$FF	reserved

The following tables define in detail the data formats for each `SystemUseID`. The MSB–LSB notation (MSB = most significant byte, LSB = least significant byte) means that the MSB occupies the lowest address, and LSB–MSB means that the LSB occupies the lowest address.

■ **Table 3-3** SystemUseID 01, ProDOS

SystemUse offset	Contents
\$00–\$01	\$42 41 (AppleSignature)
\$02	\$01 (SystemUseID)
\$03	ProDOS file type
\$04–\$05	ProDOS aux type (LSB–MSB)

■ **Table 3-4** SystemUseID 02, HFS

SystemUse offset	Contents
\$00–\$01	\$42 41 (AppleSignature)
\$02	\$02 (SystemUseID)
\$03–\$06	HFS fileType (MSB–LSB)
\$07–\$0A	HFS fileCreator (MSB–LSB)
\$0B	\$00 (padding for even length)

■ **Table 3-5** SystemUseID 03, HFS: bundle bit set

SystemUse offset	Contents
\$00-\$01	\$42 41 (AppleSignature)
\$02	\$03 (SystemUseID)
\$03-\$06	HFS fileType (MSB-LSB)
\$07-\$0A	HFS fileCreator (MSB-LSB)
\$0B	\$00 (padding for even length)

■ **Table 3-6** SystemUseID 04, HFS

SystemUse offset	Contents
\$00-\$01	\$42 41 (AppleSignature)
\$02	\$04 (SystemUseID)
\$03-\$06	HFS fileType (MSB-LSB)
\$07-\$0A	HFS fileCreator (MSB-LSB)
\$0B-\$8A	HFS ICN# resource (MSB-LSB)
\$8B	\$00 (padding for even length)

■ **Table 3-7** SystemUseID 05, HFS: bundle bit set

SystemUse offset	Contents
\$00-\$01	\$42 41 (AppleSignature)
\$02	\$05 (SystemUseID)
\$03-\$06	HFS fileType (MSB-LSB)
\$07-\$0A	HFS fileCreator (MSB-LSB)
\$0B-\$8A	HFS ICN# resource (MSB-LSB)
\$8B	\$00 (padding for even length)

■ **Table 3-8** SystemUseID 06, HFS: bundle bit set

SystemUse offset	Contents
\$00-\$01	\$42 41 (AppleSignature)
\$02	\$06 (SystemUseID)
\$03-\$06	HFS fileType (MSB-LSB)
\$07-\$0A	HFS fileCreator (MSB-LSB)
\$0B-\$0C	HFS ICN# resource (MSB-LSB)

Authoring software can simply copy the finder flags as retrieved by the HFS call `GetFileInfo`. Only bits 5 (always `switchLaunch`), 12 (system file), 13 (bundle bit), and 15 (locked) are used. All other bits are ignored or set based on the internal working of the file system translator.

The ProDOS filename transformation

This section defines a mechanism that preserves ProDOS filename syntax. Legal filenames in ProDOS differ from legal filenames under ISO 9660 as follows:

- ProDOS filenames allow multiple periods; ISO 9660 does not.
- ISO 9660 requires that the two separators—period (.) and semicolon (;)—exist, and that the semicolon (;) be followed by a version number. This requirement is for files only, not for directories.

To preserve ProDOS filename syntax in the ISO environment that you must perform a transformation of some sort. Fortunately, this specific combination of file systems allows you to define a reversible transformation which the authoring tool and receiving system can use to hide the fact that a transformation has occurred.

When creating an ISO 9660 disc from ProDOS source files, the authoring tool must perform the following transformation on all filenames:

- Replace any periods (. , \$2E) with underscore (_ , \$5F).
- If the filename refers to a directory, the transformation is finished.
- If the filename refers to a file, append the characters .;1 (\$2E, \$3B, \$31).

Table 3-9 provides some examples of possible filename transformation for common Apple II filenames.

■ **Table 3-9** ProDOS-to-ISO 9660 filename transformation

ProDOS filename	Type	ISO 9660 filename
PRODOS	file	PRODOS.;1
BASIC.SYSTEM	file	BASIC_SYSTEM.;1
SYSTEM	directory	SYSTEM
DESK.ACCS	directory	DESK_ACCS
START.GS.OS	file	START_GS_OS.;1

- ◆ *Note:* The Volume Identifier in the Primary Volume Descriptor is a filename and, therefore, must be transformed. It must be transformed as if it were a directory name.

When the ProDOS Transformation bit is set in the Protocol Identifier, the receiving system can handle the necessary conversions such that the original ProDOS filenames can be used to refer to all files and directories on the volume.

1. Before searching the disc for a given pathname, perform the transformation described above.
2. Before returning any names found, reverse the transformation above:
 1. Strip the characters .;1 (\$2E, \$3B, \$31) if present.
 2. Replace any underscores (_ , \$5F) with periods (. , \$2E).

This process must be done for every file and directory on disc.

Guidelines for transforming HFS filenames

Because most HFS filenames are illegal in the ISO 9660 specification, some transformation will be required when you are creating an ISO 9660 disc with HFS files. No reversible transformation is possible without degrading performance. Therefore, to help produce more consistent discs, publishers and authoring tool publishers should use the following guidelines for when performing the transformation:

- Convert all lowercase characters to uppercase.
- Replace every illegal character, including periods, with an underscore (_ , \$5F).
- If the filename must be shortened, truncate the rightmost characters.
- If the filename refers to a file, append the characters ;1 (\$2E, \$3B, \$31).

ISO 9660 associated files

Associated files are exactly analogous to resource forks. Though the format of associated files is clear in the ISO 9660 specification, it is restated here.

An associated file is defined as having the associated bit set in the file flags byte of the directory record. It has exactly the same file identifier as its counterpart, and resides immediately before its counterpart in the directory. The associated file is treated as the resource fork; its counterpart is treated as the data fork of the file.

For example, if a file FOO.;1 has an associated file, there will be two adjacent directory records named FOO.;1—the first one (the resource fork) will have the associated bit set, and the second one (the data fork) will have the associated bit clear.

Chapter 4 **HyperCard and CD-ROM**

This chapter introduces possible ways of using HyperCard as a retrieval engine for or interface to Macintosh CD-ROM products. It also provides a brief high-level overview of HyperCard concepts. The information presented in this chapter is not tutorial in nature and requires that you have been exposed to, and understand, HyperCard. For a more comprehensive description of HyperCard and the HyperTalk language, see the *HyperCard Script Language Guide*, an Apple Technical Library book published by Addison-Wesley.

Chapter 5 contains information about capturing sound for integration in HyperCard applications and working with sound on the Macintosh. ■

A look at HyperCard for CD-ROM

Just as CD-ROM is an ideal method to distribute vast amounts of diverse data, HyperCard is an excellent way to organize and display this information.

Perhaps the single most interesting advantage HyperCard offers you as a developer is the opportunity to present users with a consistent interface—something currently lacking in the CD-ROM world. And, because HyperCard is included with all Macintosh computers, it's an interface familiar and available to a large customer base.

Another development advantage is HyperCard's easily programmable graphic Macintosh interface—one that utilizes all of the toolbox routines. HyperCard allows you to construct your Macintosh interface more quickly than if you programmed it in Pascal or C. Hence, your software development time and cost will be reduced significantly.

For example, if you choose to use HyperCard as an interface, you can put the majority of your effort into the content and functionality of your database. The user can work with the familiar point-and-click environment rather than a series of complicated command lines and function keys.

Best of all, HyperCard is flexible and extendible enough to suit a variety of users.

In addition, your existing retrieval and information products can be inexpensively moved into the HyperCard environment in one of two ways: (1) by putting your information into HyperCard stacks and using your talents and existing software to produce an innovative, easy-to-use retrieval engine for HyperCard, or (2) by porting the internals of your retrieval software over to the Macintosh, then utilizing the HyperCard external command feature (described later) and graphic interface to access your information in its present format.

Even without using HyperCard extensions or creating a new retrieval engine, HyperCard provides an excellent platform from which to publish on a CD-ROM. HyperCard stacks, which provide the user with buttons and easy-to-follow links to your data, can be put on CD-ROM. HyperCard then accesses the stacks on the CD-ROM and presents your unchanged data to the user.

HyperTalk: The HyperCard scripting language

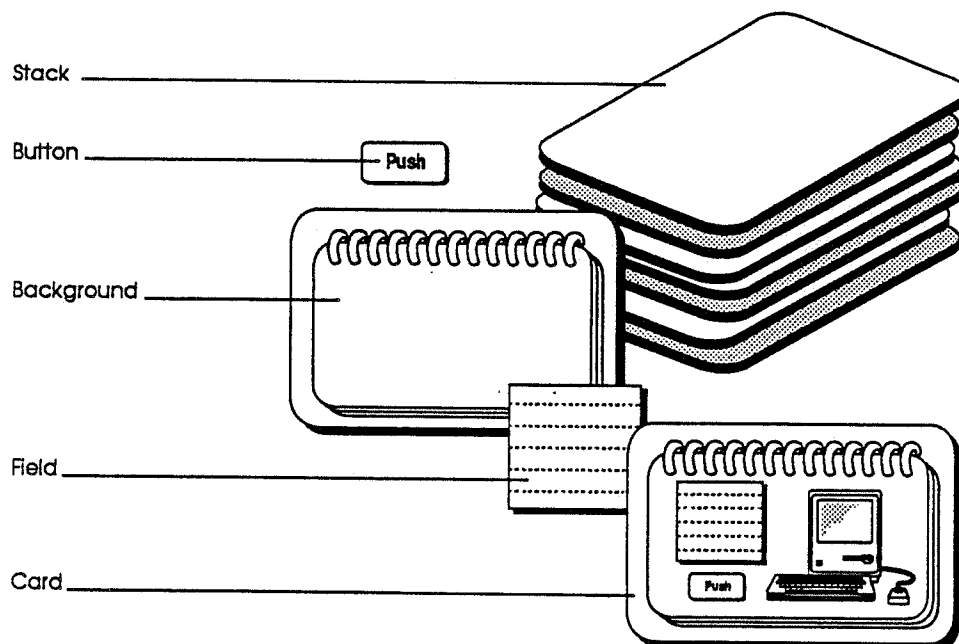
HyperTalk™ is the powerful, English-like scripting language of HyperCard, providing the tools to build the visual and interactive retrieval models possible with HyperCard. It activates HyperCard and is the glue you use to link the various parts of your stack together.

You use HyperTalk to send messages to and from HyperCard objects. A message can be generated by a click of the mouse, the opening of a card, a statement typed into the Message box, or some other event. How a given object responds to a particular message depends on its *script*, and all HyperCard scripts are written in HyperTalk.

HyperCard objects

There are five kinds of objects in HyperCard: buttons, fields, cards, backgrounds, and stacks. (Examples of HyperCard objects are shown in Figure 4-1.) The basic unit of information is the card. When you look at the screen of a Macintosh computer running HyperCard, what you see foremost is a card. Each card is associated with a background, and a background may be (and usually is) shared by more than one card.

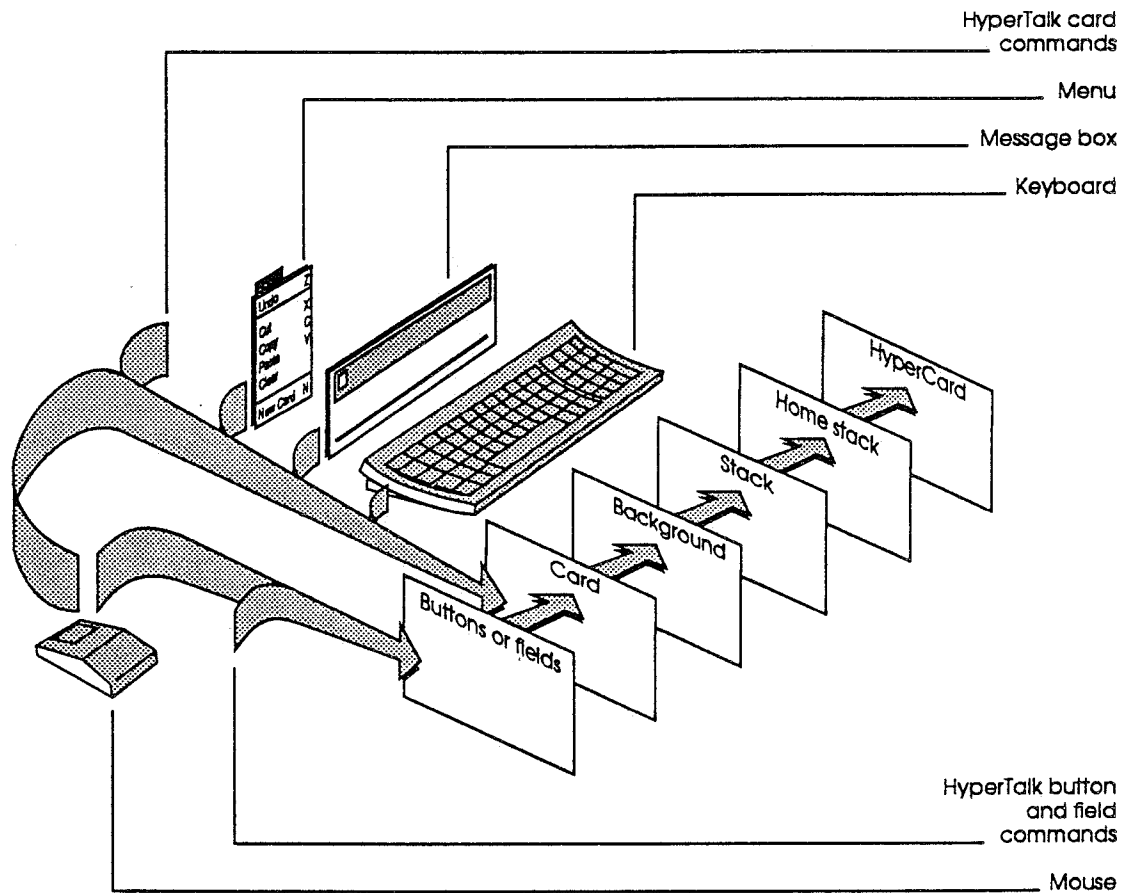
■ **Figure 4-1** HyperCard objects



Cards and backgrounds are both the size of the card window, which is the size of the original Macintosh screen. Cards overlay the background; what you see in the card window belongs to the card, or, where the card is transparent, to the background. Backgrounds are like a basic template for a card. For example, a stack of index cards have a common background—either lined or plain—and what you write on them distinguishes the individual cards. The card and background can both contain graphics to enhance the appearance and draw attention to areas of interest. The graphics can be generated with HyperCard's built-in Paint tools or pasted in from other sources.

Cards and backgrounds are grouped in stacks, which are the Macintosh files that contain your information. The layers of objects that make up what you see in HyperCard are shown in Figure 4-2.

■ **Figure 4-2** The layers of HyperCard objects on cards and backgrounds



Cards and backgrounds also contain buttons and fields. Buttons are action objects or “hot spots” on the screen—when you click a button with the Browse tool, something usually happens. For example, clicking a button can take you to the next card in a stack. Fields carry editable text: strings of characters. The Browse tool hand pointer changes to an I-beam when it’s positioned over a field of unlocked text, allowing you to manipulate the text with it. All editable text in HyperCard is carried in fields. (There may also be characters on the card or background that are Paint text. Such characters are not editable once they are placed; they are part of the picture on the card or background.)

HyperCard as a data-retrieval engine

This section provides an overview of two HyperCard retrieval engine scenarios for CD-ROM information access. The first scenario covers designing a CD-ROM entirely with HyperCard, and the second scenario describes working with an existing CD-ROM designed initially for the non-Macintosh environment.

Designing a CD-ROM retrieval engine with HyperCard

If your CD-ROM contains large amounts of text, two important pieces of software need to be written for retrieval software that uses HyperCard stacks. The first piece of software is the inversion (or data analysis) software, which you use to build an index of your stack in an auxiliary table. The inversion software can be a simple or complex piece of work, depending on how many ways you want to allow the user to search for an item. The contents of a typical HyperCard stack auxiliary table would allow you to answer these queries: Given a list of words, what cards contain these words? Given a list of cards, what words do these cards contain?

The inversion software is run after you have built your stack, and are ready to freeze it and put it on CD-ROM. To use the inversion engine, you must first write a HyperCard script, using HyperTalk, that effectively goes through the entire stack—every card, every field, every word—and calls the inversion engine, which records every word in the stack and where it occurs. The result obtained by the inversion software is put into the auxiliary table. For example, if the inversion engine were recording the word *Apple* and found 200 occurrences in the stack, it would build an auxiliary table containing word descriptor information, such as that the first occurrence of Apple is on card number 2, background field 3, word offset 100, and so on. The data structure for the word descriptor might be as follows:

Card ID number = 32 bits

Background ID number = 32 bits

Background/card field flag = 1 bit

Field ID number = 7 bits

Word offset in card field < 16 bits

When the inversion process is completed, and the index in the auxiliary table is built, the table is stored along with the stack to be placed on the CD-ROM. (See Figure 4-3.) Because the read-only text on the disc cannot be modified in any way, the table is placed on the CD-ROM and remains a static data structure.

Here is an example HyperCard script that extracts all of the words and field names from a HyperCard stack and puts them into a plain text file.

```
function cleanName thisName
-- strip quotes from field or bkgrnd name
    delete first char of thisName
    delete last char of thisName
    return thisName
end cleanName

on emit entity,cardNum,bkgndID,cardID,bkCd,fieldID,wdOffset
-- write one word & descriptor vector to the output file
    global outFile
    put cardNum & tab & bkgndID & tab & cardID & tab & bkCd & tab &
    fieldID & tab & wdOffset & tab & entity into outString
-- log to the message box for progress indicator
    put outString
-- replace linefeed with return for non-A/UX™ usage
    put linefeed after outString
    write outString to file outFile
end emit

function okChar theChar
-- check to see if char type is alphanumeric
    if theChar is empty then return true
    if theChar > "/" and theChar < ":" then return true
    if theChar > "@" and theChar < "[" then return true
    if theChar > "_" and theChar < "(" then return true
    return false
end okChar
```

```

function stripWord theWord
-- remove nonalphanumerics from the ends of word
  repeat while not okChar(first char of theWord)
    delete first char of theWord
  end repeat
  repeat while not okChar(last char of theWord)
    delete last char of theWord
  end repeat
  return theWord
end stripWord

on invert
  global outFile
  ask "Output file?"
  if it is empty then exit invert
  put it into outFile
  open file outFile
  put number of last bkgnd into lastBkgnd
-- loop over all bkgnd types
  repeat with thisBkgnd = 1 to lastBkgnd
    go to next card of bkgnd thisBkgnd
    put the id of this bkgnd into thisIdent
    put the name of this bkgnd into thisName
    -- record bkgnd name
    if word 3 of thisName is empty then
      emit "&" & cleanName(word 2 of thisName),0,thisIdent,0,0,0,0
    end if
    put number of bkgnd fields into endField
    -- record bkgnd field names
    repeat with thisField = 1 to endField
      put the id of bkgnd field thisField into fieldIdent
      put the name of bkgnd field thisField into fieldName
      if word 4 of fieldName is empty then
        emit "*" & cleanName(word 3 of fieldName),0,thisIdent,0,0,-
        fieldIdent,0
      end if
    -- end field repeat
  end repeat
end repeat

```

```

put the id of this card into cardIdent
put cardIdent into endIdent
-- loop over all cards in bkgnd
repeat
  put number of this card into thisNum
  -- loop over bkgnd fields in card
  repeat with thisField = 1 to endField
    put the id of bkgnd field thisField into fieldIdent
    put number of words in bkgnd field thisField into endWord
    -- loop over words in field
    repeat with thisWord = 1 to endWord
      put stripWord(word thisWord of bkgnd field thisField) into theWord
      if theWord is empty then next repeat
      emit theWord,thisNum,thisIdent,word 3 of cardIdent,0,-
        fieldIdent,thisWord
    end repeat
  end repeat
-- loop over card fields, if any
put number of card fields into endCField
repeat with thisField = 1 to endCField
  put the id of card field thisField into fieldIdent
  put the name of card field thisField into fieldName
  -- check if named
  if word 4 of fieldName is empty then
    emit "" & cleanName(word 3 of fieldName),thisNum,-
      thisIdent,word 3 of cardIdent,1,fieldIdent,0
  end if
  -- loop over card field words
  put number of words in card field thisField into endWord
  repeat with thisWord = 1 to endWord
    put stripWord(word thisWord of card field thisField) into theWord
    if theWord is empty then next repeat
    emit theWord,thisNum,thisIdent,thisNum,1,fieldIdent,thisWord
  end repeat
-- end card field repeat
end repeat
-- end card repeat

```

```

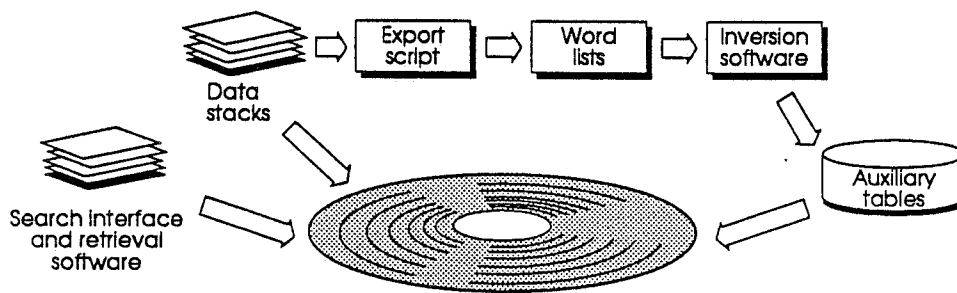
    go to next card of bkgnd thisBkgnd
    put the id of this card into cardIdent
    if cardIdent is endIdent then exit repeat
  end repeat
  -- end background repeat
end repeat
end invert

```

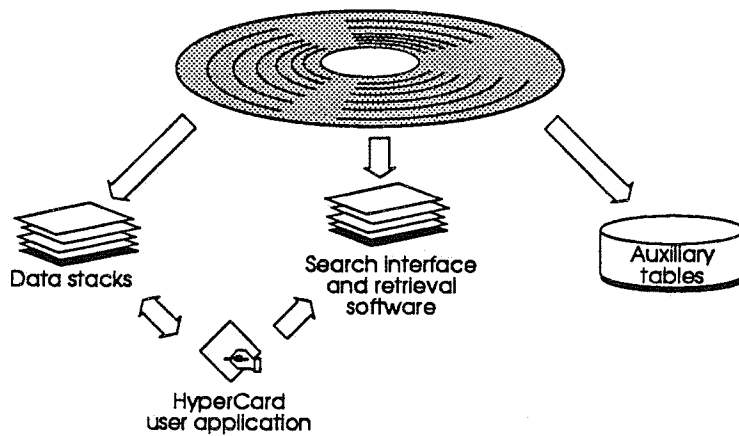
The output from this script could be used as input to your batch mode inversion software. Alternately, you could modify this script to directly call your inversion software, which is coded as a HyperCard XCMD. (For more information about HyperCard XCMDs, see the *HyperCard Script Language Guide*.)

The second piece of code that must be written is the run-time retrieval software. The retrieval software interrogates the auxiliary tables, based on the particular set of words, and retrieves the desired information. This is a full-text retrieval model, which essentially bypasses the Find mechanism of HyperCard and hands back a list of cards by ID, which contain the words that match the search criteria. The pieces that make up the HyperCard CD-ROM are shown in Figures 4-3 and 4-4.

■ **Figure 4-3** Pieces of a HyperCard-based CD-ROM (data preparation)



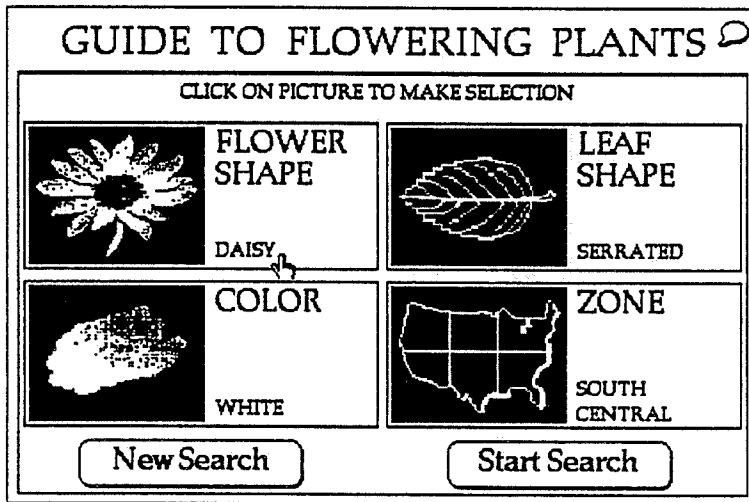
■ **Figure 4-4** Pieces of a HyperCard-based CD-ROM (data playback)



A friendly interface

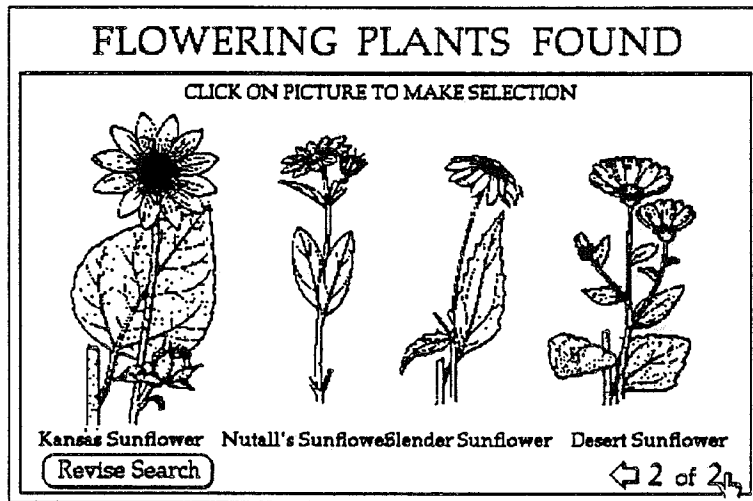
The type of ASCII search interface that provides only lists of words or cards containing the words would be far too crude to present to the user. However, HyperCard can be used to enhance the user's experience. You can use HyperCard to quickly create a graphical interface that provides the user with fields and buttons that access the underlying search commands. The interface should be simple, consistent, modifiable (if the search interface is on disk rather than on the CD-ROM), and appropriate for the kind of data you are presenting. For example, to present a catalog of information, you might build a card with an array of buttons and fields like those shown in Figure 4-5.

■ **Figure 4-5** Example of HyperCard interface



In this example, the user opens the stack to find a group of related textual and graphic elements. These elements help to guide the user through the information. The user clicks the various elements to make up the type of flower or plant he or she wants information about. Once all of the elements describing the plant have been selected, the user clicks the Search button. The Search button script calls the retrieval software, which returns a list of plants that satisfy the search criteria supplied with the descriptive elements. The plant list could be used to call a card that is further enhanced with graphic elements, such as pictures of the plants, as shown in Figure 4-6. However, it might simply be used to fill in a card field that allows the user to click one of the plant names to get more information about the selected plant.

- **Figure 4-6** Example of graphic elements that enhance the data presentation



Working with an existing CD-ROM

In this scenario, a library of information already exists on a CD-ROM. The existing CD-ROM has been prepared on a mainframe for use on a non-Macintosh computer. The CD-ROM uses a non-Macintosh retrieval engine and interface, which may include the use of command lines or function keys.

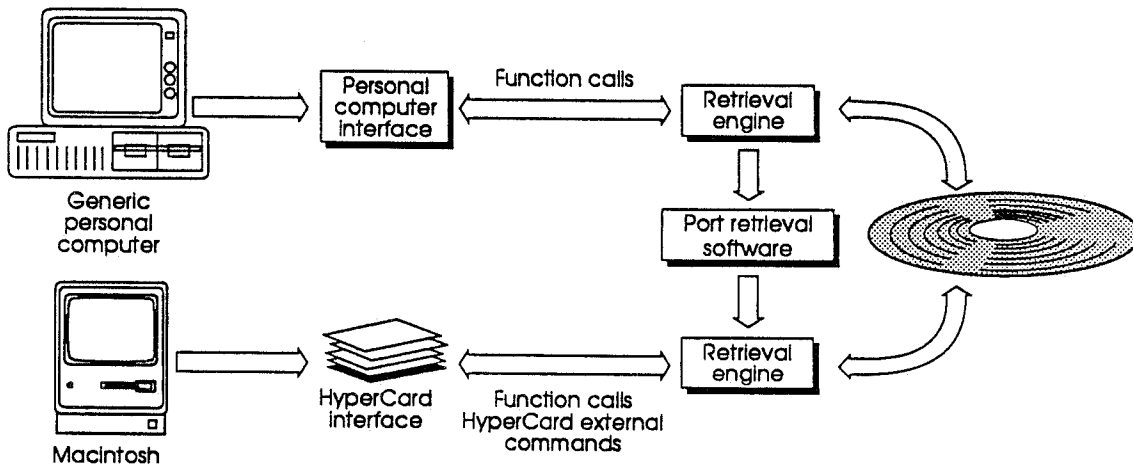
Most CD-ROM developers and information providers leverage their software investment by writing the retrieval engine in a portable language, such as C or Pascal, so that it can be moved into other operating environments. The user interface portion of the code, however, may not be portable. Therefore, one of the problems in the past that kept information providers from developing a CD-ROM delivery system for the Macintosh has been the great expense of porting the user interface to the Macintosh.

Because the non-Macintosh interface may not be very portable (command lines and function keys aren't standard on the Macintosh), and many programmers are not familiar with the Macintosh programming environment, it may take three months or more just to understand how to implement the interface. This is where the HyperCard scenario, described below, can save both time and money.

HyperCard provides a complete set of tools to compose the human interface and customize the retrieval model to fit the needs of your application. Engineering development time can be cut approximately in half by using HyperCard as a user interface construction kit.

First, take the existing CD-ROM retrieval engine and port it to the Macintosh operating environment. If the job is done right, there should be a clean software interface layer consisting of function calls to the retrieval engine. HyperCard external commands (XCMDs) are written so that HyperTalk scripts can call these retrieval functions. Then you use HyperCard to build the graphic Macintosh interface with the proper hooks for the XCMDs. The block diagram in Figure 4-7 shows the steps described for this HyperCard scenario.

■ **Figure 4-7** Preparing an existing non-Macintosh CD-ROM for the Macintosh environment



The HyperCard interface for existing CD-ROM databases

HyperCard is an excellent tool for quickly building an interface to your information. For example, if you want to work with an existing CD-ROM that contains a text-based card catalog for a library, you might use a HyperCard interface that includes buttons and fields like those shown in Figure 4-8. The richly graphic interface will improve the use, appearance, and marketability of your product.

■ **Figure 4-8** Sample card for text search

BOOK SEARCH

Type in the information you know about Author, Title or Subject

Author: Sa

Title:

Subject:

Number of items found:

restart

Sabatelli, Wayne
Sabbath, Steve
Sabbatini, Charles A.
Sackett, Richard
Sahm, Ralph
Sahota, Vicki
Sakaguchi, Sandra
Salaiz, Ken
Salazar, Kristina
Sanchez, David W.
Sandler, Jim
Sanford, Bill
Santora, Greg

The buttons and fields provide the user with an intuitive way to supply the search criteria. Instead of typing in some arcane string of commands, the user would click on the Author, Title, or Subject field, and start to type. When the user starts typing in the Author field, a list of available authors appears. The user can then select the author he or she wants from the list, and a list of corresponding titles appears, as shown in Figure 4-9.

■ **Figure 4-9** Book titles corresponding to author's name

BOOK SEARCH Type in the information you know about Author, Title or Subject

Author:

Title:

Subject:

Number of items found: **11**

- A Traveler's Guide to Spain by Ken Salaiz
- A Visitor's Guide to France by Ken Salaiz
- Art Museums in France by Ken Salaiz
- English Country Inns by Ken Salaiz
- France on a budget by Ken Salaiz
- Quick trips to Europe by Ken Salaiz

If the user desires more information about one of the titles, he or she can simply click on the corresponding title. More detailed information appears, as shown in Figure 4-10.

■ **Figure 4-10** Detailed information in catalog card

Author: Ken Salaiz

Title: A Visitor's Guide to France

Published: John Wiley & Sons: New York, NY: 1983

Topics: travel
foreign countries

Catalog Number: 877.49 C453n

2 of 11

Once you've ported the retrieval engine and created the corresponding XCMD interface, the user interface shown above is relatively easy to build with HyperCard. The user makes information queries by using familiar HyperCard interface conventions, and the results are found by the XCMDs and put into HyperCard fields that correspond to the information being requested. Each field can have a unique look according to its properties. Fields can be shadowed, scrollable, plain rectangles, transparent, and opaque. Fields can contain any font that is available in the user's System file.

- ◆ **Note:** If you plan to use fonts that are not in the standard release of the Apple Macintosh System file, you will have to license and supply the unique font resources with your product.

More information about fields is available in the *HyperCard Script Language Guide* and the *HyperCard User's Guide*. One of the best ways to learn about the possible uses for HyperCard and how it will enhance your CD-ROM product on the Macintosh is hands-on experimentation.

Additional interface considerations

The information that follows is a brief set of guidelines for interface design on Apple computers. The contents of *Human Interface Guidelines: The Apple Desktop Interface*, published by Addison-Wesley, provides more detailed information about the rationale behind the guidelines given here.

The human interface puts the power of the computer in the user's hands. It is the communication link between the user and the information contained by the machine. It's what presents the information to the user and accepts information from the user.

Make your interface consistent; there is no need to reinvent the interface for each application. A consistent interface ensures ease of use, prevents the tools from keeping the user from the task, and makes mistakes less likely.

Think about your interface from the user's point of view. Users aren't trying to use computers—they're trying to get a job done. The user is instinctively curious. Let users explore without making mistakes. Users want control of their environment. People are productive, effective, and imaginative when the environment is enjoyable and challenging.

The ten general principles

The following ten principles can be applied to any successful interface design.

- metaphors from the real world
- direct manipulation
- see-and-point
- consistency
- WYSIWYG
- user control
- feedback and dialog
- forgiveness
- perceived stability
- aesthetic integrity

Metaphors from the real world: They are quickly understood and take advantage of people's direct experiences with their immediate world.

Direct manipulation: Users want to feel that they are in charge of the computer's activities. People expect their physical actions to have physical reactions. The Macintosh control panel allows the user to make changes to the Macintosh and receive immediate results. For example, the user can change the desktop pattern by clicking pixels on or off.

See-and-point: Recognition, not recall, is the key here; see-and-point rather than remember-and-type. Most users are not programmers and are not familiar with using command-line interfaces. Users prefer to select actions from alternatives presented on the screen. For example, users should be able to pick quickly from a list of graphical options. In the Flowering Plants example shown in Figure 4-5, the user clicks the icon representing the basic shape of the flower, and doesn't need to use any specific keywords.

Consistency: Effective applications are consistent within themselves and consistent with one another. Consistency allows users to transfer a general set of skills learned from one application to another. For example, users expect that all Apple applications will consistently have Apple, File, and Edit menus, and that items such as Save will always be found in the File menu. HyperCard stacks always have a return arrow button, and that button always means "take me back to the previous place I was at."

WYSIWYG: What you see on the screen is what you get in a printed version. No secrets from the user, no abstract commands without immediate feedback. For example, making a copy of the document "Bonzo" results in a visual representation of the copy, appropriately named "Copy of Bonzo."

User control: The user, not the computer, initiates and controls all actions. In other words, the user acts and the computer reacts. In possibly disastrous situations the computer provides warnings, but the user makes the decision whether to proceed with an action or not. For example, a user is free to move any item to the Trash. However, if an application is put into the Trash, a warning dialog box is presented before any action to destroy the application is taken. Here again, the user is given the choice to make the final decision.

Feedback and dialog: Always keep the user informed about what is taking place. Provide immediate feedback with visual effects, sound, and/or brief and direct messages expressed in the user's vocabulary, not the programmer's. Reduce complex activities into small simple steps. Feedback can be as shown in a number of ways, from letting the user know the mouse click was received, to showing the watch to indicate a delay while processing, to using a dialog box to show progress of the current search or other activity.

Forgiveness: Users learn best through exploration. If possible, don't let users make mistakes; always provide them with a way out. Actions should always be reversible. Inform users when an action can't be completed or is not available. Provide escape routes from potentially dangerous actions. For example, when a user selects Erase Disk from the Special menu, a dialog box always asks whether the user wants to proceed or to cancel the process.

Perceived stability: Users are most comfortable in an environment that appears to stay the same. Use consistent graphical elements and visual landmarks. Unavailable items aren't mysteriously missing—they're dimmed. For example, not all menu items are available at all times. However, all menu items are always shown, with the unavailable options in gray. Users are never mystified as to why an item has disappeared.

Aesthetic integrity: Use attractive displays, and express visual clarity, simplicity, and consistency. Visually confusing displays detract from the effectiveness of any product. Different areas of information should have distinct and different appearances. Use the skills of a professional graphic designer, when possible.

Programming

Your programming strategy should incorporate several concepts that also follow the general interface guidelines presented earlier.

- User options and computer actions should appear modeless. A given action always has the same results, regardless of past activities.
- Use an event loop that allows the user to do anything at any time.
- Make actions reversible—always provide a way out.
- The screen is central—it depicts all human/computer interactions.
- Use plain language—communicate in concise, simple terms that are direct and unambiguous. Use the skills of a professional writer or editor, when possible.

Layout and design

The layout and design of your product will be greatly enhanced if you follow the suggestions listed below.

- Use both text and graphics for expressing ideas.
- Use graphics to communicate, not just impress the users. Use graphics to provide users with a clear context of where they are and where they can go.
- Integrate graphics early on in the design process.
- Keep graphics simple and straightforward.
- Avoid using too much text, and break up large areas of text for ease of readability.

HyperCard interface design

The following suggestions apply to designing interfaces for HyperCard-based CD-ROM products.

- Convey the central idea of a card through its title and background environment.
- Make sure the user is given the tools to get around. In other words, can the user find things and return from an exploratory session in a stack?
- Make text-entry areas obvious.
- Use graphics in the physical appearance of buttons to make them obvious and their function readily understandable.

Testing your interface design

The key to any good application for a CD-ROM product is making sure the interface does the job it was intended to do. User testing is generally the best sounding board. Here is a list of good guidelines for testing your interface.

- Try out rough ideas on several users early in the design process. Waiting for a fully functional prototype can be a costly mistake.
- Schedule testing in a regular cycle of feedback and revision.
- Use naive testers.
- Have the users talk and describe what they are doing as they work.
- Videotape users as they work so that you have a permanent record that you can use to review their reactions and see where to improve on any faulty areas.

Localization—international issues

Providing a product for the international market or designing multilanguage versions of a product requires great expertise. Many of the design methods used for preparing a product for the international market are described in *Inside Macintosh*, the *Human Interface Guidelines*, and *Software Development for International Markets*. A few basic guidelines are presented here.

- Use icons rather than text whenever possible. Use culturally neutral graphics, or create alternative graphics for different cultures.
- Keyboards vary from country to country. Be sure any required keystrokes are easily performed on any keyboard.
- Units of measure such as inches and centimeters, and formats for time and date, differ from country to country. Try to isolate this kind of information in resources.
- Foreign text usually takes up more space than English, so leave extra room. Don't create programs that rely on strings' having a particular length, or on specified ASCII values.
- When you're using HyperCard, use regular text in fields rather than Paint text (created with the Paint Text tool) for easy replacement. Use fonts that come with the system.
- In non-HyperCard applications, store user-visible text (dialogs, menus, and so on) in Macintosh resources where it can be easily modified, not in the program code itself.

Limitations of the HyperCard search mechanism

There is a price to pay for the built-in functionality of HyperCard. It allows easy interface prototype development, but does not provide the fastest and most flexible search mechanism for CD-ROM. The HyperCard Find function works well for single stacks with card fields not exceeding 1000 characters each, and groups of smaller stacks. However, when your information begins reaching the total storage capacity possible with CD-ROM, the Find function may slow down considerably.

HyperCard's current Find function provides right truncation, exact match, and Boolean AND search functionality. Right truncation allows the user to find words by supplying only a letter (preferably a few letters) of the word they are looking for. AND searches find two items (item and item), but do not perform proximity checking. The words can be contained anywhere on a card. HyperCard's Find function currently does not allow for searches with the Boolean operators OR and NOT.

The feedback provided by the Find function is not adequate if you require document list hits per item or hits per document.

Increasing HyperCard's performance for CD-ROM

If you prefer the HyperCard interface and have large amounts of textual information, you may consider writing your own search software as an XCMD, or porting one of the existing third-party search engines for the Macintosh.

If you are using HyperCard as your interface and search software, several methods can be used to increase the speed of the Find function.

- Split very large amounts of information among several stacks.
- Keep the number of characters on each card field under 3200. This limit prevents false hits and increases HyperCard's search performance.
- Use HyperCard version 1.2 or higher. Performance and robustness in handling large stacks has been greatly improved over that in older versions of HyperCard. To take advantage of the performance improvements, compact older stacks twice with versions 1.2 or higher.

Hints for creating HyperCard stacks for large databases

- HyperCard guarantees uniqueness of card ID numbers, not names. If you are building your own auxiliary tables, include the card ID numbers.
- Buttons should reference cards by ID, not by name, because you could have instances of the same name in multiple backgrounds or cards.
- If reference by the card name is desired, the full name space must be recorded. Include the background and card names in the auxiliary tables index. (See "Designing a CD-ROM Retrieval Engine With HyperCard," earlier in this chapter, for an explanation of auxiliary tables.)
- Make sure the volume is locked during testing; otherwise the stack could be altered during the testing process.

HyperCard stacks on a CD-ROM and locked media

On locked stacks, such as CD-ROM, HyperCard 1.2 distinguishes between user actions and those of HyperTalk scripts. Although users are prevented from making changes to CD-ROM stacks, scripts can perform most operations.

Several users may read a stack on CD-ROM set for read-only access or locked by the Finder. No special commands are required to read a locked stack. However, you should keep two things in mind when creating stacks that will be placed on CD-ROM: All changes generated by scripts, such as showing a pop-up field, disappear when any move is made to another card. For example, if you have a field that pops up with help for the user, you can't rely on the behavior of the locked stack to hide the field when the user is through with it.

Users generally can't type into fields on locked stacks, and most interfaces that allow word and text string searches require typed input from the user. The global property `userModify` allows users to modify text-entry fields for providing search criteria. If you plan to deliver your product with your search interface stack and all of the data stacks on the CD-ROM, set `userModify` to true for buttons, fields, cards, stacks, or other HyperCard objects where you need the property. If you want to save the information the user entered, store it in a global variable.

Cards in a CD-ROM stack can be printed in the usual way. If a card image is temporarily altered by a script or user, the image displayed on the screen will be printed when Print Card is selected from the File menu, not the original card on the disc. The command Print Stack rotates the cards in the stack while generating the print job; therefore, any changes to a specific card will be discarded before printing.

Operations that can't be performed on CD-ROM

The following list describes operations that can't be done by either users or scripts on CD-ROM and locked media.

- Make new cards.
- Delete, cut, or paste cards. (You can copy and paste cards to a media that can be written to.)
- Make new backgrounds.
- Compact a stack.
- Delete a stack.
- Edit patterns.
- Change the name of a stack.
- Change any scripts.
- Sort the stack.

BLANK

Chapter 5 **AppleCD SC and Sound**

This chapter examines the differences between stereo CD-Audio and processor sound for CD-ROM and describes how to integrate sound into applications to enhance other CD-ROM data. Hints for producing CD-ROM applications with CD-Audio sound are included to help guide you through the process. This chapter also provides a source listing for a sample program that makes CD-Audio calls to the Macintosh Apple CD-ROM driver. ■

CD-Audio and processor sound for CD-ROM

There are several differences between CD-Audio and processor sound. CD-Audio is the standard used to create the audio CDs you play on CD players. CD-Audio has a unique format for CDs and must be played through the speaker channels on the AppleCD SC. Processor sound is captured in files and must be played back by the host Macintosh or Apple II computer.

CD-Audio is said to produce the best stereo sound quality of any recording medium. Using stereo sound with graphics and text data would provide a high-quality multimedia presentation. However, there is a drawback involved in trying to mix CD-ROM data and audio tracks on the same CD-ROM disc: You can experience a loss of performance in your application. The CD-ROM drive not only has to seek to a specific area on the disc for the CD-Audio tracks, but also has to switch between CD-ROM and CD-Audio modes of operation. Switching modes can take up to one second, and a long seek from CD-ROM data to the requested CD-Audio track could take another half second. The user would hear and possibly see the change in application performance.

In contrast, processor sound digitized on the Macintosh and placed on CD-ROM can be accessed much faster, because the drive doesn't have to make a long seek to the CD-Audio area of the disc, and because the sound is stored as another file in the Macintosh or Apple II format. Additionally, if the sound is digitized at sampling rates comparable to CD-Audio sampling rates, the quality of the sound would be acceptable for most applications. (Sound sampling is described later in this chapter.)

This is not to say that CD-Audio doesn't have uses on CD-ROM. As a simple example, your application could be an instructional piece on the sounds of musical instruments. The user could select a period in the history of music and choose from a group of instruments for that period. Once the instrument was selected, the user might be presented a picture with some descriptive text about the selected instrument and hear the sound of the instrument played in true stereo CD-Audio. The sound would be played after the graphics and text are displayed on screen, creating the impression of a natural transition in the program. Table 5-1 compares the differences between processor and CD-Audio sound programming requirements and CD-ROM application performance.

■ **Table 5-1** Comparison of processor and CD-Audio sound

Processor sound	CD-Audio sound
Uses less disc space than CD-Audio (dependent on sampling rate).	Uses more disc space than processor sound.
Cheaper to capture sound.	More expensive to capture sound.
Allows variable sampling rates. (Highest possible sampling rate is lower than CD-Audio.)	Standard sampling rate is higher than processor sound sampling rates.
Lower sound quality than CD-Audio.	Best sound quality.
All Apple computers have built-in sound hardware to play processor sound.	Requires additional speakers to play CD-Audio.
Easier to capture sound.	Requires studio time and equipment. Media format must conform to CD vendor specification.
Sound is captured in Apple file format.	Plays through CD-ROM drive and doesn't use computer RAM.
Uses computer RAM space during capture and playback. Sound segment has to be loaded into memory.	
More difficult to program complex sound segments such as multiple voices and continuous sound.	Simply send an <code>APLAY</code> command to the CD-ROM drive. See <code>APLAY</code> command description in Chapter 7, and the program sample later in this chapter.
Uses CPU cycles that slow down program execution on the Macintosh Plus, Macintosh SE, Apple IIe, and Apple IIGS. Better performance on the Macintosh II.	Does not use CPU cycles.
Faster to start playback if sound is preloaded into RAM.	Takes longer to start playback—seek to audio track can take 1 to 2 seconds.
If sound segment is smaller than available RAM, you can preload the sound in memory and play it back while accessing other data.	Does not allow access of data tracks while playing an audio track.

■ **Table 5-1** Comparison of processor and CD-Audio sound (continued)

Processor sound	CD-Audio sound
On the Macintosh Plus and Macintosh SE performance is slower while playing and loading large continuous sound segments. The Macintosh II provides better performance while playing and loading sound.	Does not use CPU time or computer RAM space.
Testing programs is easier and less expensive because sounds are standard files rather than CD-Audio tracks.	You need to press a CD-ROM to really test the relationship between your program and the CD-Audio.

Using processor sound in applications

Sound is an important piece of any multimedia application designed to work with the AppleCD SC. The combination of Macintosh and HyperCard provides an inexpensive hardware and software development environment for multimedia CD-ROM projects. The mixture of sound, graphics, and text will certainly be the ideal way for many of the future training and educational CD-ROM applications to present instructional ideas and information. Industrial training applications might use HyperCard and sound to add life to training diagrams, textual instructions, and parts lists. Educational "HyperCard films" can be threaded through reference material, using the attractive power of animation, speech, and music to draw students' attention to graphic and textual content that illustrates the class curriculum.

The following section describes methods for getting processor sound into the Macintosh for use with HyperCard multimedia applications. (See Chapter 4 for a discussion of HyperCard as retrieval software and the user interface.)

Processor sound on the Macintosh and Apple II

Sound is captured and stored in computers such as the Macintosh and the Apple II through a technique called *digitizing* or *sampling*. Sampled sound is the same type of sound used on compact discs and sampling electronic keyboards. (However, there are differences between the sampling rates for processor sound and compact disc audio, which affect sound quality.) During the digitizing process, the original sound waveform is examined (or sampled) many times per second. The sound wave is recorded and stored in the computer, producing a sound data file. (Several products are available for capturing sound, and are discussed later in this chapter.) The computer recreates the digitized sound by fetching the data from the sound file at the same sampling rate at which it was recorded. The computer then drives a loudspeaker to the recorded amplitude for each instant the sound was sampled, thus producing a synthesized version of the original sound waveform.

Any digitized sound has two distinct and important characteristics: the frequency at which it was sampled, and the precision with which the sound level was recorded. The best-quality sound from a Macintosh Plus, Macintosh SE, or Macintosh II is digitized and played back at 22,000 samples per second (or 22 kHz) with a precision of one part in 256, corresponding to 8 bits per sample. Thus, sound sampled at the 22 kHz rate will consume 22K of memory space during the sampling process for each second of playing time. Compare this to CD-Audio, which samples at 44 kHz with 16-bit precision, producing 176K per second of playing time for the two stereo channels. The greater the sampling frequency and the greater the precision, the more memory you will use.

Processor sound is considerably less expensive to capture. Computer sound sampling hardware is inexpensive and the software allows you to capture the sounds in a readily usable file format. However, this inexpensive initial cost can be offset by the potential difficulty you may encounter while programming. Sounds use up space on disc and in RAM. Capturing large sound segments requires a lot of RAM, and playing back large sound files requires that you take great care in managing your memory space.

When playing back sound segments larger than available memory, you need to fetch the sound from disc and load portions of the sound into RAM while playing the sound. On the Macintosh Plus, Macintosh SE, and Apple II, program performance slows and it is difficult or impossible to access other program data while fetching the sound data from the disc. If the sound segment does not exceed memory, you can preload the entire sound into RAM, which allows access of other data on the disc while the sound is being played. On the Macintosh II, continuous play of large sound segments is an easier task because of the Macintosh II's memory management capabilities.

Using CD-Audio sound in applications

The expense of capturing and preparing CD-Audio for a CD-ROM applications is justified by the quality of the sound and the relative ease of programming your application for sound. The AppleCD SC plays sound independently from the computer, which means you don't have to use up CPU time or load the sound into computer RAM. You only need to issue a play command from your application and the AppleCD SC does all the work.

Some limitations to application performance exist when using CD-Audio sound.

- You cannot access other data on the disc while playing an audio track. If you do try to access other data, the disc drive stops playing the audio track and moves to the data area of the disc. Be aware of this fact and program accordingly.
- Testing the performance of your application and the relationship between your data and audio information requires that you press a CD-ROM. The next section gives you hints to help you increase program performance, and decrease the need to have multiple CD-ROM test discs pressed, when using CD-Audio sound.

Tips for developing a CD-ROM application with sound

This section includes tips for CD-ROM application developers and managers of CD-ROM projects that will include both sound and data information. The following guidelines will be helpful in planning for the build and test portions of your CD-ROM project.

The build environment

The integration of program and CD-Audio information is performed by the CD-ROM publisher that creates your disc. This process is called *authoring* the information for CD-ROM. A master tape is created during the CD-ROM authoring process. The master tape includes both the program data and CD-Audio information. CD-ROM publishers require that you deliver your CD-ROM content in a format that can be readily used. Generally, CD-Audio sound is delivered to the publisher in a tape format called PCM 501K. The program data can be delivered on a large hard disk, CD-WORM, or on tape. Check with CD-ROM publishers for the specifications they require.

Sound studios record CD-Audio on PCM 501K tapes. These studios can also convert sound recorded on less expensive equipment, such as stereo tape players, camcorders, and so on, to digital format. Note that a simple rule applies for this conversion process: the more money you spend, the greater the level of sound quality.

The master tapes created during the authoring process contain information required by the portion of your program code that controls audio playback from the AppleCD SC. You need to get a listing of the master tape from the CD-ROM publisher to keep track of the information. This information comprises the title of the track (either data or audio), the track numbers, the duration of the data and audio tracks, and the time from the beginning of the CD-ROM to the beginning of the data or audio track. (A sample of a master tape listing is shown in Figure 5-1.) Given this information, you can code your program to play back the audio in one of three methods: absolute time, time from the beginning of a track, or track number. (See the `APLAY` command description in Chapter 7.)

■ **Figure 5-1** A sample CD-ROM master tape listing

TITLE: Your CD-ROM					TAPE NO.				
ARTIST: Your Company									
TITLE	TNO	X	E M P	Duration M S F	Time Code H M S F	CD Time M S CF			
Data	1	0 1	N N	PT: 2.00 B: 0.00 M: 29.58.00	2.00.00.00 2.00.02.00	.00 2.00			
This could be you . . .	2	0 1	Y Y	PT: 2.00 B: 0.00 M: 15.09	2.30.00.00 2.30.02.00	30.00.00 30.02.00			
Sounds of Hawaii . . .	3	0 1	Y Y	PT: 2.00 B: 0.00 M: 9.00	2.30.17.09 2.30.17.10	30.17.22 30.19.25			
The tropical paradise . . .	4	0 1	Y Y	PT: 2.00 B: 0.00 M: 8.11	2.30.21.10 2.30.23.10	30.28.25 30.30.25			

It is possible to create a CD-ROM that contains all of the sound information for your application in one track. However, because the arrangement of data on the CD-ROM is such that the data track is always track 1 and CD-Audio is in the following track or tracks, the task of locating separate sounds in one track becomes very difficult. When the sounds are combined in one track, your program plays back the sounds according to either the offsets in minutes-seconds-frames from the beginning of the track, or the offsets in absolute minutes-seconds-frames from the beginning of the CD-ROM. Each time you add a sound segment to your application, the offsets of the sounds within the track change, and each time you change the data portion of your application, the absolute time in minutes-seconds-frames from the beginning of the disc changes. To keep track of the sound offsets for your program, a new master tape listing is required each time you make a data or sound change.

You can make your programming task easier by keeping each sound segment in a separate track. With the sound segments in separate tracks, you only need to get the track number to play a sound. Because the track numbers for your sound segments will remain the same throughout the project, you will not have to spend time updating your program code every time the data portion of your application changes.

- ◆ *Note:* You can decrease the time to seek to an audio track if the master tape is authored with minimum time gaps between sound tracks. This time can be customized by your CD-ROM publisher to suit your program needs. The standard gap size is around 2 seconds. This gap time can be reduced to 0.5 seconds.

The test environment

The information in this section can help you during the test phase of any CD-ROM application that combines data and CD-Audio.

- The more integrated your data and audio, the longer the test cycle will last.
- CD-ROM is a read-only medium. To help cut down CD-ROM publishing costs, do your testing on a hard disk that can be write-protected. A write-protected hard disk provides a reasonable simulation of CD-ROM performance.
- Running your program data on a hard disk and accessing a CD for the audio data do not accurately produce the same kinds of errors that may exist if the program and audio data are both on the CD-ROM. Plan to do more testing once you have pressed a CD-ROM with both the your program and audio data.
- The quality (error rate) of CD-ROM discs varies between CD-ROM publishers. You should do testing with CD-ROM discs from more than one vendor. This will give you the opportunity to make a vendor selection based on the highest-quality CD-ROM.

HyperCard and sound

HyperCard's HyperTalk language allows you to build tools for the integration of sound into Macintosh HyperCard applications. HyperTalk scripting and XCMDs can extend processor sound excerpts into synchronized sound and graphics films. HyperCard films achieve a feel similar to that of a video production, but have some distinct advantages over conventional projected audiovisual media. HyperCard films can be played straight through or be interactively controlled by the user, so that he or she can pause during the film and branch to other related material, or continue at any time. You can use HyperCard scripts to build presentations and training materials, to create narrated tours of HyperCard databases on CD-ROM, or to do something as creative as design talking directory pages for a telephone-book stack.

Both processor and CD-Audio sound can be incorporated into the HyperCard application environment. HyperCard XCMDs for playing CD-Audio tracks on CD-ROM discs in an AppleCD SC allow you to develop HyperCard applications that use high-quality voice, sound effects, and music. Several inexpensive third-party sound digitizers make it possible for you to sample any sound you can create and save it to a standard file format or HyperCard 'snd' resource format. The sampled sounds can be easily incorporated into HyperCard applications.

- ◆ *Note:* Applications should use either processor sound or CD-Audio sound, but not both in the same application. Combining both types of sound can cause human interface problems. Users may find it difficult to tell where the sound is coming from and not have their playback system set up to take advantage of the sound in the application.

Storing sound for use with HyperCard

HyperCard stores sampled sounds in the resource fork of a stack file. The resource type is 'snd '. (The space after the letter *d* is required; resource type names must have four characters.) Each sound is given a unique name within the HyperCard stack.

HyperCard can deal with sound sampled at four different rates: 22 kHz, 11 kHz, 7.4 kHz, or 5.5 kHz. Lower sampling rates create fewer data points and use less memory for equivalent playing time of a higher sampling rate. Using lower sampling rates lets you fit more sound in memory and on discs. Keeping the amount of data to be transferred between the disc and memory at a minimum is important when implementing continuous play in a HyperCard stack. However, low sampling rates do not reproduce high-pitched sounds.

A phenomenon associated with sampling is that sound with a frequency greater than one-half the sampling rate is not reproduced accurately. For example, a sampling rate of 5.5 kHz results in inaccurate reproduction of sounds above 2.75 kHz, which is the middle of the third octave above middle C. Thus, 5.5 kHz is not an adequate sampling rate for music, although it might serve for intelligible voice narration. The halfway point for a given sampling rate is called the Nyquist frequency. A sampling rate of 22 kHz (Nyquist frequency equal to 11 kHz) is adequate for most popular music and compares well to the fidelity of AM radio, but the sound reproduction is not near the quality of CD-Audio sound.

An incoming pitch above the Nyquist frequency will be falsely played back, because it is shifted down during sampling. For example, a steady tone at 5.6 kHz would sound like a low hum at 100 kHz when sampled at 11 kHz. This effect is called *aliasing*. With sampled music, aliasing creates an odd buzz or chirp where a high pitch should be present. To avoid it, you must remove the high-frequency part of the audio with a low-pass filter or a graphic equalizer before sampling.

Sound-capturing hardware

The minimal system configuration for capturing and manipulating Macintosh sound is a Macintosh Plus with a SCSI hard disk. (The SCSI hard disk isn't mandatory, but it's faster than the Hard Disk 20, and longer segments of sound play back without breaking up.) To get the sound into the computer of your choice, you need an audio digitizer, a peripheral device that performs the actual sampling of an analog signal under control of the computer. Several models are available. Check with music stores and computer dealers for price and features of sound digitizers, or read reviews in magazines dedicated to sound production for CD-Audio and computer-aided music composition.

Some of the features you should look for in a digitizer are these:

- Sampling at all four Macintosh rates: 22, 11, 7.5, and 5.5 kHz.
- Deletion of higher-frequency sounds before digitizing. If this feature is not available on the digitizer, you will need to connect a graphic equalizer or low-pass filter in the audio line ahead of the digitizer to avoid aliasing.
- An RS-232 interface connector for computer control. The interface should plug directly into the newer Macintosh computers with the 8-pin mini-circular RS-232 serial ports and require no external power. If these features are not available, the digitizer will require a 5-volt power supply and cable interface adapter.
- A jack for audio input with a microphone or external source, such as a tape recorder.

The same feature set applies to sound digitizers for the Apple IIe and Apple IIGS; however, on the Apple IIGS you can sample sounds at rates up to 26 kHz.

Sound production software

Most digitizers come with sound production and editing software. The software should provide the ability to play the incoming sound in soundwave mode for adjusting the sound level and should allow capturing segments of digitized sound up to the limits of your computer's memory. The software sound editor should let you look at the sound wave in several resolutions so that you can select and modify sections with point-by-point editing. Additional features for adding effects such as amplification, ramp up or down, and optional effects like reverb and flange will also be useful for sound enhancement and customization. For the Macintosh, the software should store sound in a recognizable file format, and possibly provide file conversion to HyperCard stack 'snd' resource format.

Additional software

If you are using sound production software that doesn't provide HyperCard file conversion, you can use a public domain stack called SoundCapConvert, written by Bill Atkinson. The SoundCapConvert stack takes the files produced by many of the popular digitizing packages and inserts them into a stack in the required 'snd' format. You can obtain this stack from most HyperCard user groups, bulletin boards, or on-line special interest groups (SIGs). A copy of the ResEdit™ program is also helpful for checking which sounds are present in a stack, and moving or deleting them.

Controlling the AppleCD SC audio features

One of the features of the AppleCD SC is the capability to operate in the CD-Audio mode. As described earlier, the CD-Audio mode allows you to play audio CDs on the AppleCD SC, as you would on the a CD player. However, unlike a stereo CD player, the AppleCD SC CD-Audio mode is controlled only through software commands initiated from a host computer. This section provides a commented code sample for playing a CD-Audio track on a CD-Audio disc.

Software written to use the AppleCD SC CD-Audio control and status calls controls the AppleCD SC through the SCSI Manager and AppleCD SC device driver on the Macintosh, and through the SCSI SmartPort interface on the Apple IIe and Apple IIGS.

- ◆ *Note:* To ensure AppleCD SC application compatibility in the future, always make the SCSI control and status calls through the Macintosh AppleCD SC device driver or through the Apple II SmartPort, rather than going directly to the device.

The AppleCD SC audio control and status calls for the Macintosh and Apple II SmartPort are listed separately in Chapter 7, "AppleCD SC Software."

- ◆ *Note:* You can also control the CD-Audio capability of the AppleCD SC by using HyperCard XCMDs to make audio calls directly through the AppleCD SC device driver.

The source code for a small Macintosh application program, `Play.c`, is provided on the following pages. This sample program allows a user to play a CD-Audio track by clicking an audio track file icon. `Play.c` shows how to make CD-Audio calls to the AppleCD SC device driver, and does not include many of the features commonly associated with larger Macintosh applications. You must have the Audio CD Access and Foreign File Access system files in the System Folder in order to see audio track file icons when a CD-Audio disc is opened.

The program `Play.c` was written in LightSpeed C (a product of Symantec Corporation) and must be compiled along with its associated resource `Play.r`.

```

/* File:      Play.c
 * Purpose:   demonstrate AppleCD SC control
 * Date:     17 June 1988
 * Comments:
 *           *This code is written for LightSpeed C version 2.15.
 *           *Requires some associated resources (in the file Play.r).
 *           *With this program on your hard disk, you can click
 *           *on a CD-Audio track and the AppleCD SC drive will
 *           *start audio play, beginning at that track and continuing
 *           *to the end of the disc.
 */

#include <MacTypes.h>
#include <QuickDraw.h>
#include <WindowMgr.h>
#include <Color.h>
#include <ColorToolbox.h>
#include <EventMgr.h>
#include <SegmentLdr.h>
#include <FileMgr.h>
#include <HFS.h>
#include <StdFilePkg.h>
#include <Pascal.h>
#include <DialogMgr.h>

#include <stdio.h>

/* Some constants, from the AppleCD SC Developer's Guide Chapter 7 */
#define TRACKADDR 2 /*specify csParam +0 track number address format*/
#define STEREO 9 /*specify BCD value for csParam +8 stereo play
mode*/

/* Some commands, from the AppleCD SC Developer's Guide Chapter 7 */
#define APLAY 104 /*specify the APlay command csCode*/
#define ASTOP 106 /*specify the AStop command csCode*/
#define ASTATUS 107 /*specify the AStatus command csCode*/

#define errorID 128 /* The resource ID for the error dialog */

/* prototype function declarations. */
void Error(char *, short);
void ExtractTrackNo(char *, short *);
OSErr AStop(short, short);
OSErr APlay(short, short);
OSErr Play(char *, short);
short GetDrvRefNum(short);
void PlayOnStartup(short);
Boolean AskAndPlay(void);
void main(void);

```

```

/*****
*
* Function: Error
*
* Purpose: Report an error to the user.
*
* Returns: Nothing
*
* Side Effects: None
*
* Description: Takes a string and a result code. Convert the result
*              into a pascal string using NumToString() and then
*              just do a case switch to determine which friendly error
*              message to put up on the screen. ParamText() sets up
*              the alert so that it has the correct information.
*              Use Alert() to actually display the message to the
*              user.
*
*****/

```

```

void
Error(where, result)
char *where;
short result;
{
    DialogPtr    dPtr;
    short        itemhit;
    Str255        strResult;
    long          longResult;

    longResult = result;
    NumToString(longResult, strResult);

    switch (result)
    {
        case -56:
            ParamText(where, strResult, "\pSpecified drive number
                        doesn't match any number in the drive
                        queue.", "\p");
            break;
        case -51:
            ParamText(where, strResult, "\pReference number specifies
                        nonexistent access path.", "\p");
            break;
        case -50:
            ParamText(where, strResult, "\pError in parameter list.",
                        "\p");
            break;
        case -43:
            Par

```

```

amText(where, strResult, "\pFile not found.", "\p");
    break;
case -36:
    ParamText(where, strResult, "\pI/O error.", "\p");
    break;
case -35:
    ParamText(where, strResult, "\pNo such volume.", "\p");
    break;
case -21:
    ParamText(where, strResult, "\pBad unit number.",
"\p");
    break;
default:
    ParamText(where, strResult, "\p", "\p");
    break;
}

```

```

Alert(errorID, (ProcPtr) NULL);
}

```

```

/*****
*
* Function: ExtractTrackNo
*
* Purpose: Extract track number in BCD from PString
*
* Returns: Nothing
*
* Side Effects: *Track gets a new value
*
* Description: Extract track number in BCD from PString "name".
* "name" is always of the form "Track XX", where XX
* ranges from " 1" (1 with a leading space) to "99"
* (You can't have more than 99 tracks on a compact
* disc, if your CD conforms to the "Yellow Book"
* specification of compact disc encoding.)
*
* The track number value for the APlay and AStop
* csParam +2 is calculated by ExtractTracNo.
*
*****/

```

```

void
ExtractTrackNo(name, track)
char      *name;
short     *track;
{
    short   size;
    short   t;

```

```

    t = 0;
    size = *name;
    if (name[size-1] != ' ')
        t = (name[size-1] - '0') * 16;
    t += name[size] - '0';

    *track = t;
}

/*****
 *
 * Function: AStop
 *
 * Purpose: Stop playing an audio track
 *
 * Returns: OSErr. Probably either
 *           noErr           everything's OK!
 *           paramErr        you messed up the call somehow
 *
 * Side Effects: Stops play
 *
 * Description: The track that you pass in is the LAST track that you
 *               want to play. This means that if you wanted to
 *               play only one track, you'd pass the same value to
 *               this routine and APlay().
 *
 *               Note that this routine isn't called in this sample,
 *               but it's included for your enjoyment.
 *
 *****/
OSErr
AStop(track, refNum)
short track;
short refNum;
{
    struct {
        short    addrFormat;
        long     address;
    } csParam;

    csParam.addrFormat = TRACKADDR;
    csParam.address = track;
    return (Control(refNum, ASTOP, &csParam));
}

```

```

/*****
*
* Function: APlay
*
* Purpose: Start playing an audio track
*
* Returns: OSErr. Probably either
*           noErr          everything's OK!
*           paramErr       you messed up the call somehow
*
* Side Effects: Starts play. By default, this will play until the
*               end of the disc.
*
* Description: The track that you pass in is the first track that you
*               want to have play.
*
*****/
OSErr
APlay(track, refNum)
short    track;
short    refNum;
{
    struct {
        short    addrFormat;
        long     address;
        short    stopAddress;
        short    playMode;
    }    csParam;

    csParam.addrFormat = TRACKADDR;    /* specify track number address format */
    csParam.address = track;    /* must be in BCD */
    csParam.stopAddress = 0;    /* stop address flag */
    csParam.playMode = STEREO;
    return (Control(refNum, APLAY, &csParam));
}

```

```

/*****
*
* Function: Play
*
* Purpose: Play an audio track, given it's filename
*
* Returns: OsErr and whatever APlay() return
*
* Side Effects: Calls APlay to play the audio track
*
* Description: The two parameters passed in are the file name (which
*              will always be of the form "Track XX", XX varying from
*              " 1" to "99") and the driver reference number. Extract
*              the track number and use it to call APlay().
*
*****/
OSErr
Play(name, volume)
char      *name;
short     volume;
{
    short    trackno;
    OSErr    result;

    ExtractTrackNo(name, &trackno);
    return( APlay(trackno, volume) );
}

```

```

/*****
*
* Function: GetDrvRefNum
*
* Purpose: Get the driver reference number
*
* Returns: Short (The driver reference number)
*
* Side Effects: None
*
* Description: PBHGetVInfo() will retrieve the driver reference
*              number, given the vRefNum associated with a file.
*              We acquired the vRefNum as something passed into
*              the program, either by SFGetFile() or by
*              GetAppFile().
*
*              We'll use the driver reference number in all our
*              control calls to the driver.
*
*****/
short
GetDrvRefNum(vRefNum)
short vRefNum;
{
    HParamBlockRec io;

    io.volumeParam.ioCompletion = NULL;
    io.volumeParam.ioNamePtr = NULL;
    io.volumeParam.ioVRefNum = vRefNum;
    io.volumeParam.ioVolIndex = 0;
    PBHGetVInfo(&io, false);
    return io.volumeParam.ioVRefNum;
}

```

```

/*****
*
* Function: PlayOnStartup
*
* Purpose: Play the files passed in to our application
*
* Returns: Nothing
*
* Side Effects: Plays some files.
*
* Description: Loop, using GetAppFiles() to get the next "file"
               (actually a CD-Audio track) to play. Continue until
               we run out of files to play or until we get some
               error while trying to play a track.
*
               As this routine is currently written, it doesn't
               handle opening multiple tracks at once. (For example,
               the user selects five tracks and opens them all.) Right
               now, you'll only hear the last track (each Play()
               command cancels the previous one).
*
               Add a routine to check the status of the CD SC
               drive (using AStatus) so that you wait for a track
               to finish before starting the next track.
*
*****/
void
PlayOnStartup(count)
short count;
{
    short i;
    AppFile app;
    OSErr result;

    for (i = 0; i++<count; )
    {
        GetAppFiles(i, &app);
        if (app.fType == 'trak')
        {
            result = Play((char *)app.fName, GetDrvRefNum(app.vRefNum));
            if (result != noErr)
                Error("\pPlay returned", result);
        }
    }
}

```

```

/*****
*
* Function: AskAndPlay
*
* Purpose: Prompt the user for a track and play it
*
* Returns: Boolean. Either:
*           true   the user wants us to continue
*           false  the user wants us to exit, or an error
*                  occurred
*
* Side Effects: Play an audio track on a CD-Audio disc
*
* Description: Use SFGGetFile() with a specific type ('trak') to get
*              a list of just CD-Audio tracks. If the user selects
*              an audio track, play it.
*****/
Boolean
AskAndPlay()
{
    SFReply sf;
    short refnum;
    static Point where = {100,100};
    static OSType sft1 = 'trak';
    Boolean result;
    OSErr errorCode;

    result = true;
    FlushEvents(everyEvent-diskMask,0);
    SFGGetFile(where, "\pSelect Track to Play", (ProcPtr)NULL,
               1, &sft1, (ProcPtr)NULL, &sf);
    if (!sf.good)
        result = false;
    else
    {
        errorCode = Play((char *)sf.fName, GetDrvRefNum(sf.vRefNum));
        if (errorCode != noErr)
        {
            Error("\pPlay returned", result);
            result = false;
        }
    }
    return result;
}

```

```

/*****
*
* Function: Main
*
* Purpose: This is where we start.
*
* Returns: Nothing.
*
* Side Effects: Does whatever you tell it.
*
* Description: Do the standard Macintosh initialization. Following
* all the boilerplate, use CountAppFiles() to see how
* we were started. We could be started one of two
* ways:
* 1) The user opened our application. We will have
* a count of zero. Ask the user for a track to play.
* 2) The user clicked a CD-Audio track "file". We
* will have a nonzero count. Go play the track(s)
* that the user clicked.
*
*****/
void
main()
{
    int      message, count;
    short    refNum;

    MaxApplZone();
    MoreMasters();

    InitGraf(&thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(0);

    InitCursor();
    FlushEvents(everyEvent-diskMask, 0);

    CountAppFiles(&message, &count);
    if (count)
        PlayOnStartup(count);
    else
        while (AskAndPlay())
            ; /* null loop */
}

```

If you are planning to enter and compile this code, the resource code that follows must be a separate file named `Play.r`.

```
#include "Types.r"

/* The BNDL describes how to get to our ICN# resources
 * so that the Finder can display the icon, and tell
 * which application to open when a file of the associated type has
 * been opened.
 */

resource 'BNDL' (128) {
    'aucd',
    0,
    {
        /* array TypeArray: 2 elements */
        /* [1] */
        'ICN#',
        {
            /* array IDArray: 2 elements */
            /* [1] */
            0, 128,
            /* [2] */
            1, 129
        },
        /* [2] */
        'FREF',
        {
            /* array IDArray: 2 elements */
            /* [1] */
            0, 128,
            /* [2] */
            1, 129
        }
    }
};

resource 'FREF' (128) {
    'APPL',
    0,
    ""
};

resource 'FREF' (129) {
    'trak',
    1,
    ""
};
```

```

/* The program icon. This is what the programmer looks like,
 * roughly. Thanks to the HyperCard
 * icon stack for the inspiration. (Ever notice how everybody
 * looks pretty much the same at 32 bits by 32 bits?)
 */

```

```

resource 'ICN#' (128) {
    {      /* array: 2 elements */
        /* [1] */
        $"000F F000 003F FC00 006F 7E00 01FD F700"
        $"03D3 C380 03B4 03C0 0780 03C0 0700 00E0"
        $"0680 00E0 0680 00E0 0700 00E0 06FC 3FE0"
        $"0FBF F6E0 0B2A 4A90 0B02 40D0 0584 21A0"
        $"0578 1EA0 0500 01A0 0283 41C0 03C3 A340"
        $"01E0 0A80 00AF DD80 00AD F700 00FC 0900"
        $"006B F700 0079 DD00 005F 7900 004E D980"
        $"0081 F0C0 0300 0030 1C00 001C E000 0007",
        /* [2] */
        $"000F F000 003F FC00 007F FE00 01FF FF00"
        $"03FF FF80 03FF FFC0 07FF FFC0 07FF FFE0"
        $"07FF FFE0 07FF FFE0 07FF FFE0 07FF FFE0"
        $"0FFF FFE0 0FFF FFF0 0FFF FFF0 07FF FFE0"
        $"07FF FFE0 07FF FFE0 03FF FFC0 03FF FFC0"
        $"01FF FF80 00FF FF80 00FF FF00 00FF FF00"
        $"007F FF00 007F FD00 005F F900 004F F980"
        $"0081 F0C0 0300 0030 1C00 001C E000 0007"
    }
};

```

```

/* The ICN# for the documents "created" by this
 * application. Actually, the Audio CD Access
 * file returns this icon to the Finder. I must
 * supply this icon as well, so that if somebody
 * copies a CD-Audio track to a hard disk, it will
 * still show up with the proper icon. (Now there's
 * a useless operation-copy an audio track to
 * a hard disk.)
 */

```

```

resource 'ICN#' (129) {
    {
        /* array: 2 elements */
        /* [1] */
        $"FFFF E000 8000 3000 8200 2800 8200 2400"
        $"8300 2200 8280 2100 8240 3F80 8240 0080"
        $"8240 0080 8247 FFFC 8288 3F82 9E10 D561"
        $"BE11 AAB1 BE13 5559 9C16 AAAD 8015 5F55"
        $"801A B1AB 801D 6ED7 801A DF6B 801D 5957"
        $"801A D96B 801D 6ED7 801A B1AB 8015 5F57"
        $"8016 AAAF 8013 555D 8011 AABD 8010 D579"
        $"8010 3FF1 801F FFFF 8010 0001 FFEF FFFE",
        /* [2] */
        $"FFFF E000 FFFF F000 FFFF F800 FFFF FC00"
        $"FFFF FE00 FFFF FF00 FFFF EF80 FFFF EF80"
        $"FFFF FF80 FFFF FFFC FFFF FFFE FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
        $"FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF"
    }
};

/* Version comment. */
type 'aucd' as 'STR ';

data 'aucd' (0) {
    "Play demo 1.1"
};

```

```

/* This DITL, ALERT, ICON triplet makes up the error routine
 * dialog box. The ALERT defines the layout of the DITL and
 * ICON. The DITL contains a button marked "OK", plus three
 * spaces for ParamText() to fill in (the fourth
 * possible parameter to ParamText() is not displayed).
 */

```

```

resource 'DITL' (128, "Error", purgeable) {
    {      /* array DITLarray: 5 elements */
        /* [1] */
        {122, 226, 142, 286},
        Button {
            enabled,
            "OK"
        },
        /* [2] */
        {17, 67, 39, 286},
        StaticText {
            disabled,
            "^0"
        },
        /* [3] */
        {49, 67, 68, 108},
        StaticText {
            disabled,
            "^1"
        },
        /* [4] */
        {24, 18, 56, 50},
        Icon {
            disabled,
            128
        },
        /* [5] */
        {49, 117, 114, 286},
        StaticText {
            enabled,
            "^2"
        }
    }
};

```

```

resource 'ALRT' (128, "Error") {
    {50, 72, 202, 410},
    128,
    {
        /* array: 4 elements */
        /* [1] */
        OK, visible, silent,
        /* [2] */
        OK, visible, silent,
        /* [3] */
        OK, visible, silent,
        /* [4] */
        OK, visible, silent
    }
};

resource 'ICON' (128) {
    $"FFFF E000 8000 3000 8200 2800 8200 2400"
    $"8300 2200 8280 2100 8240 3F80 8240 0080"
    $"8240 0080 8247 FFFC 8288 3F82 9E10 D561"
    $"BE11 AAB1 BE13 5559 9C16 AAAD 8015 5F55"
    $"801A B1AB 801D 6ED7 801A DF6B 801D 5957"
    $"801A D96B 801D 6ED7 801A B1AB 8015 5F57"
    $"8016 AAAF 8013 555D 8011 AABD 8010 D579"
    $"8010 3FF1 801F FFFF 8010 0001 FFEF FFFE"
};

```

Chapter 6 **Networks and the AppleCD SC**

This chapter includes a brief overview of how to get the AppleCD SC ready for sharing information and making it available to Macintosh, Apple II, and IBM PC users on the network who have AppleShare Workstation Software. Installation and general AppleShare administrator's tasks are covered in the *AppleShare Administrator's Guide*.

You can attach one or more AppleCD SC drives to a Macintosh working as an AppleShare file server on the AppleTalk network system. CD-ROM discs formatted with Macintosh HFS, Apple II ProDOS, or High Sierra will work on the AppleTalk network as AppleShare server volumes. ■

BLANK

(Blind Folio)

Naming the CD-ROM server volume

When creating a CD-ROM specifically for use as a server volume in an AppleShare work environment, it's a good idea to give the CD-ROM a name that indicates its contents. For example, you may want the information for different work groups to be on separate volumes. You could name each volume after one of the work groups.

Whatever name you choose, be sure that each volume name is unique. Do not use the same name for more than one volume.

If only Macintosh and IBM PC users will be accessing the CD-ROM volume, use a name no more than 27 characters long. Do not use a colon (:) or begin with a period (.).

If Apple II users will be accessing the CD-ROM volume, you must use a valid ProDOS name—no more than 15 characters, including letters, numbers, and periods. You must begin with a letter. Do not use spaces in the name.

△ **Important** You cannot rename CD-ROM discs or other locked media. If a CD-ROM doesn't have a valid ProDOS name, Apple II users cannot read it on the network. Apple II users can open only those folders and files with valid ProDOS names. △

When using CD-ROM as a server volume, you must install the AppleCD SC device driver on the server startup volume and on the *Server Administration* disk. (You also need the device driver on the *Server Administration* disk so you can read the CD-ROM while preparing server volumes. A CD-ROM cannot be used as a startup volume because you cannot change its contents.)

The AppleShare server administrator can change the See Folders and See Files privileges for CD-ROMs (and other locked volumes), but only for the volume, not for the folders on the volume.

Updating the CD-ROM server volume

Many producers of CD-ROM products will be creating information that is updated and redistributed on a monthly or semiannual basis. If your CD-ROM product falls into this category, be sure you update the creation times and dates for any volumes, directories, and files to be included on the CD-ROM. This rule applies to all volumes, directories, and files, whether new or previously included on the CD-ROM being replaced.

- △ **Important** If you do not update the creation times and dates on the CD-ROM you're redistributing, the AppleShare software will not update the location of the volumes, directories, and files relative to the new disc in the AppleShare parallel directory structure. Without an updated parallel directory structure, AppleShare will not locate volumes, directories, and files on the CD-ROM being redistributed with the same names as those that were on the CD-ROM being replaced. From the AppleShare users' perspective everything will look normal, but when they try to read a file on a new CD-ROM server without updated creation times and dates, AppleShare will stop working. △

Chapter 7 **AppleCD SC Software**

This chapter describes the interaction between the AppleCD SC device driver and the Macintosh Operating System managers. It also provides the AppleCD SC system-level device control and status calls for the Macintosh device driver and the Apple II SmartPort interface. ■

BLANK

(Blind Folio)

What makes the AppleCD SC work?

The AppleCD SC device driver, which is called "Apple CD-ROM," is the device-controller software that provides a nonspecific interface to system software. The device driver responds to requests from the computer by receiving commands from the computer's operating system and converting those commands into device activity.

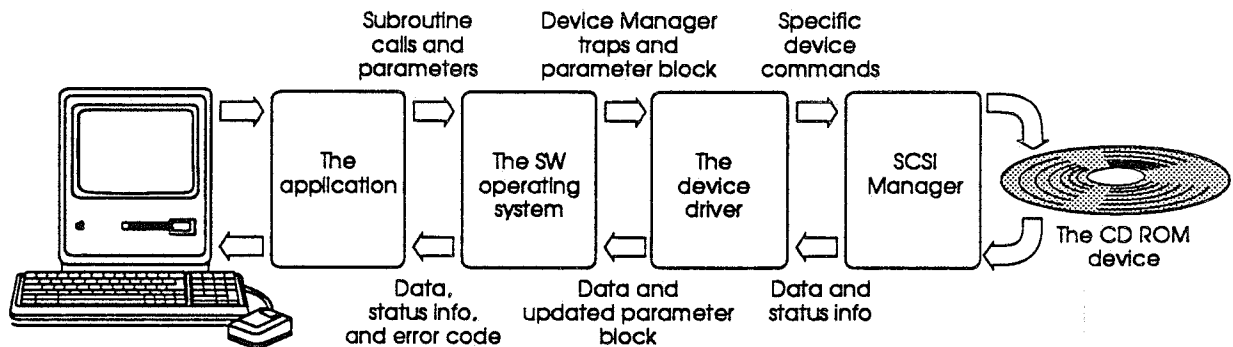
The purpose of the Apple CD-ROM device driver is to provide the software connection between Apple Macintosh computers (Macintosh Plus, Macintosh SE, and Macintosh II) and the AppleCD SC. Both HFS-compatible and non-HFS discs can be read with the driver by using standard File Manager and Device Manager procedure and function routines.

- ◆ *Note:* The Apple CD-ROM device driver provides all the functionality required by applications to read files on CD-ROM discs in the AppleCD SC drive. This functionality is available to your application without making calls directly to the device driver. File I/O is maintained through standard File Manager and Device Manager calls, such as `Read` and `SetFPos`, as described in *Inside Macintosh*, Volumes I and IV.

In addition, the AppleCD SC device driver monitors disc insertion and ejection, which gives the AppleCD SC drive the functional appearance on the desktop of the familiar 3.5-inch 800K disk. Other related activities, such as dragging the AppleCD SC icon to the Trash to unmount and eject a disc and selecting a file on an AppleCD SC, can also be done by using standard program calls such as `SFGetFile`. For more information about standard program calls such as `SFGetFile`, see *Inside Macintosh*, Volumes I and IV, and the *Programmer's Introduction to the Macintosh Family*, Chapter 7.

After the AppleCD SC device driver has been properly installed and initialized, any application can read data from a mounted AppleCD SC drive by issuing Device Manager calls. The Device Manager, part of the Macintosh Operating System, controls device behavior by sending commands to the AppleCD device driver. The AppleCD device driver, in turn, sends specific activity requests to the AppleCD SC. The relationships in the device software environment are shown in Figure 7-1.

■ **Figure 7-1** Device software environment



Controlling the AppleCD SC driver from your application

The current AppleCD SC driver installs itself when your system starts up and is readily accessible by your application. To read the drive, your application needs only to open the driver and read data through the driver.

- ◆ *Note:* All program examples in this section are written in the MPW C programming language.

To open the driver, use the Macintosh OS command that follows:

```
OSErr = OpenDriver(".AppleCD", &refNum);
```

- ◆ *Note:* For non-C programmers: The notation `&refNum` in the command example above means "address of refNum." Additionally, the period (.) in the driver name is correct and significant.

If the call is successful (`oSErr` equals `noErr`), the short integer `refNum` will contain the reference number for the driver. This number must be used in succeeding calls to read the AppleCD SC.

The AppleCD SC driver offers two ways to read sequential blocks of data from a CD-ROM drive: Macintosh OS routines and Apple CD SC SCSI commands. SCSI commands are issued to the drive directly by the application with SCSI calls, and allow several operations in addition to reading blocks of data.

Macintosh low-level routines

CD-ROM data can be read by using the following command or through the lower-level File Manager calls described later.

```
oSErr = FSRead(refNum, count, &buffPtr);
```

where `refNum` is the driver reference number, `count` is the (long) integer number of bytes to read, and `buffPtr` is a pointer to the location where the bytes are deposited. `FSRead` is a sequential read starting where the last operation left off. The File Manager call `SetFPos` may be used to change this position. (`SetFPos` sets the mark of the open file, whose access path is specified by `refNum`, to the position specified by `posMode` and `posOff`.)

Standard Macintosh result codes are negative integers or zero (zero meaning "no error"). If `oSErr = -36`, then an I/O error was encountered. If this code is returned during a disk read, execute a `Status` call:

```
oSErr = Status(refnum, csCode, &csParam);
```

where `refnum` is the value that was returned by the `OpenDriver` call, `csCode` should equal 99, and `csParam` is a 22-byte record that contains the sense-bytes describing the error condition that occurred. (For a description of the error codes returned by the driver, see the section "Macintosh AppleCD SC Driver Control and Status Calls," later in this chapter.)

The low-level routine `PBStatus` can also be used to obtain sense bytes.

Macintosh Device Manager routines

Your development environment may require you to use low-level Device Manager routines. These routines function like the related routines presented above, but require the construction of an I/O parameter block before executing the call. More information is available in both the File Manager and Device Manager chapters of *Inside Macintosh*, Volume II.

The Macintosh SCSI Manager offers two different commands for reading data from the AppleCD SC: `SCSIRead` and `SCSIRBlind`. The only difference between these two commands is the “handshaking” method used between the Macintosh and the CD-ROM drive. During a `SCSIRead` operation, handshaking occurs after each byte is transferred. When the `SCSIRBlind` command is executing, handshaking takes place only for the first byte of each block of data transferred.

The following call tells the AppleCD SC driver which read command to use:

```
OSErr = Control (refNum, csCode, &csParam);
```

where `refNum` is the value returned by the `OpenDriver` call, `csCode` should equal 78, and `csParam` is an integer Boolean variable that will be `TRUE` for `SCSIRBlind` or `FALSE` for `SCSIRead` reads.

Using the Macintosh AppleCD SC SCSI routines

If you need to exercise more control over your drive than the Macintosh OS routines offer, your application will need to execute SCSI commands directly to the drive. To use the AppleCD SC SCSI commands effectively, you should be familiar with the following:

- ANSI X3T9.2 Committee SCSI Specification, Revision 17B
- *Inside Macintosh*, Volume IV, Chapter 13, “The SCSI Manager”
- the AppleCD SC SCSI commands described later in this chapter

◆ *Note:* You still open the driver by using the Macintosh OS routine described earlier in this chapter.

The Control routine used to pass SCSI commands through the SCSI Manager to the CD-ROM drive is as follows:

```
OSErr = Control(refNum, csCode, &csParam);
```

where `csCode` must equal 77. This code tells the device driver to build and execute a SCSI command based on the data to follow. `csParam` is a block of data with the following structure:

```
struct csParam {          /* csParam structure */
char *cspComBlkAddr;      /* Address of SCSI command Block */
char *cspPPProgAddr;      /* Address of program or NIL
                           if no transfer */
char *cspBufAddr;         /* buffer address for transfer or
                           NIL if no transfer */
long cspTransSize;        /* Size of transfer in bytes or NIL
                           if no transfer */
long cspTicks;            /* Ticks to wait until completion */
char cspComBlkSize;       /* Size of the command block being
                           sent*/
}
```

The signed value of the `cspTransSize` (size of transfer in bytes) field indicates the direction of data transfer, as listed in Table 7-1.

■ **Table 7-1** Description of the signed value in the `cspTransSize` field

Signed value	Description of signed value
0	No bytes are being transferred.
+ positive	Reading data.
- negative	Writing data to the drive (actually writing parameters to the controller).

Macintosh AppleCD SC driver control and status calls

Control calls send information to the device driver, while `Status` calls request information from the driver.

The high-level `Control` and `Status` calls have a similar format. Each requires three parameters: first, `refnum` is the driver reference number returned by the `Open` call. Next, `csCode` tells the driver what type of information is being sent or requested, and last, `csParamPtr` points to the information itself. The format of the calls is:

```
OSErr TheCall (refnum, csCode, csParamPtr)
int refnum, csCode;
char *csParamPtr;
```

where `TheCall` is either a `Control` or `Status` call. It can also be seen that both calls return a result code, `OSErr`, to the calling routine.

If the application programmer chooses, low-level control and status routines can be used as well. With low-level calls, the routine parameters to be passed to the driver are contained in a parameter block. The format of the parameter block is well-defined in *Inside Macintosh*, Volumes I, II, and IV and will not be repeated here. The variables passed in the high-level calls are placed in the parameter block before the low-level call is executed. The low-level call format is:

```
OSErr ThePBCall (paramBlkPtr, asynch)
char *paramBlkPtr;
int asynch;
```

where `PBCallName` is either `PBControl` or `PBStatus`, `paramBlock` points to the parameter block, and `asynch` is always `FALSE`.

The high-level calls can be used when there is only one-way transfer of information as a result of the call. Others, like `ReadTOC`, for example, function both like a `Control` call and like a `Status` call. That is, control information is passed to the drive in the `z` area, and status information is found by the caller in the same data area after the call has been completed. High-level control and status calls cannot be used for these calls; low-level control and status routines must be used instead.

The list of Macintosh CD-ROM device driver control and status calls are presented in the following format.

Call Name (csCode)

Description: A brief statement of the purpose or function of the call.
Input: A list of the contents of `csParam` passed to the call, if any.
Output: A list of the contents of `csParam` returned by the call, if any.
Return codes: All codes that could be returned by the call.

Macintosh CD-ROM device driver control calls

This section includes the standard control calls supported by the Apple CD-ROM device driver. The audio control calls are listed in a separate section.

Eject (7)

Description: The control call `Eject` causes the AppleCD SC drive to eject the CD-ROM disc. Supported only by AppleCD SC drives.

Input: none

Return codes:

<code>NoErr</code>	No error
<code>badUnitErr</code>	Bad reference number
<code>offLinErr</code>	No disc in drive
<code>notOpenErr</code>	Driver is not open

MediaIcon (21)

Description: The control call `MediaIcon` returns a pointer to an icon list and a name string to caller. To learn how your application can use this call, see the description of `csCode 21` in the Disk Driver chapter of *Inside Macintosh*, Volume V.

Input: none

Output: `csParam +0` Address of icon list and name string

Return codes:

<code>NoErr</code>	No error
<code>badUnitErr</code>	Bad reference number
<code>unitEmptyErr</code>	Bad reference number
<code>notOpenErr</code>	Driver is not open

DriveIcon (22)

Description: The control call `DriveIcon` returns a pointer to an icon list and a name string to caller. See the description of `csCode 22` in the Disk Driver chapter of *Inside Macintosh*, Volume V.

Input: none

Output: `csParam +0` Address of icon list and name string

Return codes:

<code>NoErr</code>	No error
<code>badUnitErr</code>	Bad reference number
<code>unitEmptyErr</code>	Bad reference number
<code>notOpenErr</code>	Driver is not open

DriveInfo (23)

Description:	The control call <code>DriveInfo</code> returns information about the AppleCD SC drive's physical location, size, and other characteristics to caller. See the description of <code>csCode 23</code> in the Disk Driver chapter of <i>Inside Macintosh</i> , Volume V.	
Input:	none	
Output:	<code>csParam +0</code>	Byte 0: A return value of 1 indicates unspecified drive. Byte 1: A return value of 14 indicates secondary drive, removable, SCSI, external.
Return codes:	<code>NoErr</code>	No error
	<code>badUnitErr</code>	Bad reference number
	<code>unitEmptyErr</code>	Bad reference number
	<code>notOpenErr</code>	Driver is not open

SCSI (77)

Description:	The control call <code>SCSI</code> lets an application send a SCSI command directly to the AppleCD SC drive. If an <code>ioErr</code> error is returned, make a <code>GetSense</code> status call (described later) to retrieve the sense bytes returned to the driver from the AppleCD SC. The sense bytes may tell you more about the error that occurred during the <code>SCSI (77)</code> call.	
Input:	<code>csParam +0</code>	Address of SCSI command string
	<code>csParam +4</code>	Address of Transfer Instruction Block or NIL
	<code>csParam +8</code>	Reserved (4 bytes of 0); must be 0 values
	<code>csParam +12</code>	A 2-byte value; 1 = read and -1 = write
	<code>csParam +14</code>	Reserved (2 bytes of 0)
	<code>csParam +16</code>	Number of ticks ($\frac{1}{60}$ sec) to wait before completion phase is started
	<code>csParam +20</code>	Length of SCSI command string (word)
Return codes:	<code>NoErr</code>	No error
	<code>badUnitErr</code>	Bad reference number
	<code>unitEmptyErr</code>	Bad reference number
	<code>notOpenErr</code>	Driver is not open
	<code>ioErr</code>	Input/Output error

RSwitch (78)

Description: The control call `RSwitch` lets an application specify whether polling for new data on the SCSI bus will occur only at the beginning of blocks or for each byte during read commands. See the description of `SCSIRead` and `SCSIRBlind` in the SCSI Manager chapter of *Inside Macintosh*, Volume IV.

Input: `csParam+0` `TRUE` = polling on each block
 `FALSE` = polling on each byte (word)

Return codes: `NoErr` No error
 `badUnitErr` Bad reference number
 `unitEmptyErr` Bad reference number
 `notOpenErr` Driver is not open

VarBlk (79)

Description: The AppleCD SC drive supports variable block sizes. The control call `varBlk` sets the device block size.

Input: `csParam+0` The block size (integer). Legal block sizes in bytes are 512, 256, 1024, 2048, 2336, and 2340 for the AppleCD SC drive.

Return codes: `NoErr` No error
 `badUnitErr` Bad reference number
 `unitEmptyErr` Bad reference number
 `notOpenErr` Driver is not open
 `paramErr` Invalid block size parameter

UserEject (80)

Description: The control call `UserEject` enables or disables the eject button on the drive, as well as the SCSI `Eject` command described earlier.

Input: `csParam +0` (2 bytes)
Byte value
1: enables eject button
2: disables eject button (default)

Output: none

Return codes:

<code>NoErr</code>	No error
<code>badUnitErr</code>	Bad reference number
<code>unitEmptyErr</code>	Bad reference number
<code>notOpenErr</code>	Driver is not open
<code>controlErr</code>	Driver cannot respond to this call
<code>offLinErr</code>	No disc in drive

Macintosh CD-ROM device driver audio control calls

The following device driver calls control the CD-Audio features of the AppleCD SC drive. The audio calls described below are not required for implementing computer processor sound on CD-ROM, but are required for accessing audio data in CD-Audio tracks on CD-ROM discs and audio CD discs.

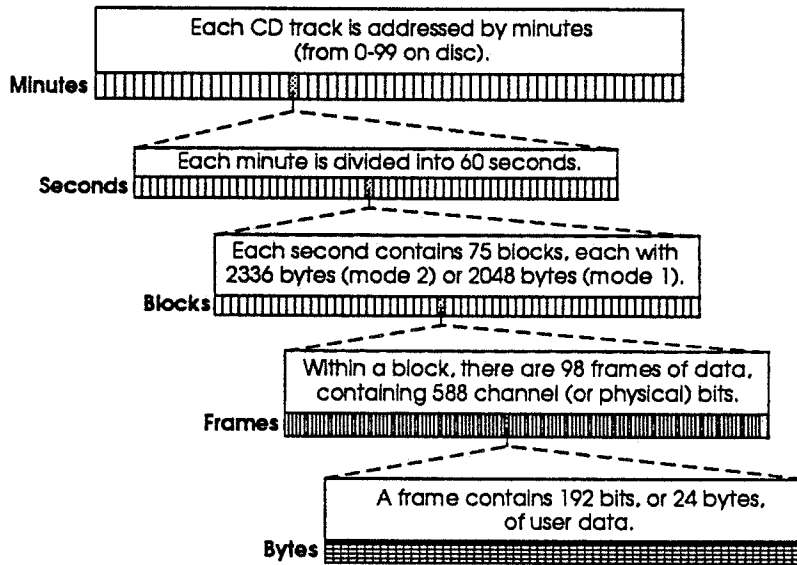
In this chapter, two sets of terms, which can easily be confused, are used when specifying the address format for certain audio control calls. These terms are *absolute minutes-seconds-frames* format and *minutes-seconds-frames* format. Absolute minutes-seconds-frames format specifies the relative running time to the address from the beginning of the disc. Minutes-seconds-frames format specifies the relative running time to the address from the beginning of a specified track.

Audio compact discs and CD-ROM discs conform to two standards, called the *Yellow Book* and the *Red Book*. The Yellow book specifies audio standards; the Red book specifies additional standards for CD-ROM. To make audio calls to the Macintosh driver, you need to understand the possible addressing formats for audio blocks on a CD. The three addressing formats are

- logical block
- absolute minute, second, and frames
- track number

An audio CD contains up to 74 minutes of sound. (By decreasing the inter-track gap and other tricks, you can put more than 74 minutes of sound on a CD. However, doing this is only marginally within the Yellow book standard. Compact discs that contain more than 74 minutes of audio may not be playable on all drives.) The data structure for audio minutes is divided into 60 seconds. Each second contains 75 blocks. Each block contains 98 frames. Each frame contains 24 bytes of data. The breakdown of this data structure is shown in Figure 7-2. The music on a CD is divided up into tracks; you can have a maximum of 99 tracks on a CD.

■ **Figure 7-2** CD-ROM data organization



The CD-ROM driver allows you to address sound down to the block level; that is, down to the $\frac{1}{75}$ th of a second. You can address sound by specifying a logical block number. For example, start playing at logical block 1,234,567 from the start of the disc. You can address sound by absolute minute, second, and frame number. For example, start playing at minute 42, second 30, frame 15 on the disc. You can address sound by track number. For example, start playing at track 2.

- ◆ *Note:* Technically the $\frac{1}{75}$ th of a second portion of the CD-ROM sound data structure is a block, as shown in Figure 7-2. However, in the control call descriptions in this chapter and throughout the CD-ROM industry the term *frame* is frequently substituted for *block* when referring to the $\frac{1}{75}$ th of a second part of the sound data structure. For example, when you address sound using the type 2 audio address format, you specify the address in minutes-seconds-frames; where frames represents $\frac{1}{75}$ th of a second rather than the technically correct $\frac{1}{98}$ th of $\frac{1}{75}$ th of a second.

The driver requires that the track number or absolute minute-second-frame numbers be specified in BCD. (The decimal number 12 is stored as hex 0x0102, not as 0x000C.)

Many audio calls return a *control field*. The control field describes the format of the current track, and has the following values:

Bits

3210	Function
00x0	2 audio channels without preemphasis
00x1	2 audio channels with preemphasis
10x0	4 audio channels without preemphasis
10x1	4 audio channels with preemphasis
01x0	Data track
01x1	Reserved
11xx	Reserved
xx0x	Digital copy prohibited
xx1x	Digital copy permitted

Many audio calls return a *play mode*. The play mode field describes how to play the audio track, and has the following values:

Bits

3210	Effect
0000	Muting on (no audio)
0001	Right channel through right channel only
0010	Left channel through right channel only
0011	Left and right channels through right channel only
0100	Right channel through left channel only
0101	Right channel through left and right channel
0110	Right channel through left channel, left channel through right channel
0111	Right channel through left channel, left and right channels through right channel
1000	Left channel through left channel only
1001	Left channel through left channel, right channel through right channel (stereo)
1010	Left channel through left and right channels
1011	Left channel through left channel, left and right channels through right channels
1100	Left and right channels through left channel
1101	Left and right channels through left channel, right channel through right channel
1110	Left and right channels through left channel, left channel through right channel
1111	Left and right channels through left channel, left and right channel through right channel (mono)

In the absolute minute-second-frame address, the ranges are as follows:

Minutes	00 to 99 (effectively, from 00 to 73)
Seconds	00 to 59
Frames	00 to 74

To issue a call to the driver, you can use either the high-level or low-level driver calls described in the Device Manager chapter of *Inside Macintosh*, Volume II.

As mentioned earlier, the high-level calls require that you pass in three parameters: the driver reference number, the `csCode` indicating which call is being made, and a pointer to the `csParam` area containing control information specific to that call. For example, to issue a type 1 `ReadTOC` call you could issue the following commands (this example is in MPW C):

```
#include <Files.h>

/* set up somewhere else by an OpenDriver() call */
short      drvRefNum;

/*
** GetFirstAndLastTrackNumber
**   requires two parameters:
**   1) a pointer to a short which will hold the first track number in BCD
**   2) a pointer to a short which will hold the last track number in BCD
**
**   fills in these two pointers with the first and last track numbers, or
**   zero if an error occurred. Returns whatever was the driver return code.
*/
OSErr
GetFirstAndLastTrackNumber(firstTrack, lastTrack)
short *firstTrack;
short *lastTrack;
{
    OSErr  result;          /* for storing result of Control() call */
    char   csParam[22];

    csParam[0] = 0;
    csParam[1] = 1;
    result = Control(drvRefNum, 100, &csParam);
    if (result == noErr)
    {
        *firstTrack = csParam[0];
        *lastTrack = csParam[1];
    }
    else
    {
        *firstTrack = 0;
        *lastTrack = 0;
    }
    return(result);
}
```

The low-level calls require that you pass in two parameters: a pointer to a parameter block and a flag indicating synchronous or asynchronous completion. For now, the driver always is synchronous, so always pass a value of false as the second parameter. You must fill in the parameter block yourself. You may find it easier to define your own custom parameter block variants for the various calls. The following example MPW C code shows how to get audio status information:

```
#include <Devices.h>

short  drvRefNum;          /* set up somewhere else */

/*
** This is a custom version of the CntrlParam parameter
** block defined on pages II-181 and II-183 or in
** Devices.h
** */
typedef struct
{
    OElemPtr      qLink;
    int            qType;
    int            ioTrap;
    Ptr            ioCmdAddr;
    ProcPtr        ioCompletion;
    OsErr          ioResult;
    StringPtr      ioNamePtr;
    int            ioVRefNum;
    int            ioRefNum;
    int            csCode;
    /* Everything below this replaces: short csParam[11] */
    unsigned char  status;
    unsigned char  play;
    unsigned char  control;
    unsigned char  minute;
    unsigned char  second;
    unsigned char  block;
    char           unused[16];          /* for the rest */
} CDCntrlParam;
```

```

/*
** GetAudioStuff
**     requires one input, the driver reference number that we
**     got by calling PBOpen().
**     Fills in six parameters with appropriate information
**     from the driver call.
**     We're assuming that only one CD player is attached.
**     To generalize this code, fill in the ioVRefNum dynamically.
*/
OSErr
GetAudioStuff(status, play, control, minute, second, frame)
unsigned char *status;
unsigned char *play;
unsigned char *control;
unsigned char *minute;
unsigned char *second;
unsigned char *frame;
{
    OSErr result;
    CDCntrlParam myPB;

    myPB.ioCompletion = 0;
    myPB.ioVRefNum = 1;          /* we're assuming the 1 volume */
    myPB.ioRefNum = drvRefNum;  /* determined elsewhere */
    myPB.csCode = 107;

    result = PBControl(&myPB, false);
    if (result == noErr)
    {
        *status = myPB.status;
        *play = myPB.play;
        *control = myPB.control;
        *minute = myPB.minute;
        *second = myPB.second;
        *frame = myPB.frame;
    }
    return(result);
}

```

ReadTOC (100)

Description: ReadTOC returns the table-of-contents information of a particular CD. There are three types of ReadTOC calls. They are distinguished by the contents of the word at csParam+0.

Input:

csParam+0 (word)	Type of ReadTOC call
0x0001	Request the first and last user tracks
0x0002	Request the lead-out (end of disc) address in absolute minute-second-frame format
0x0003	Request track starting addresses for user-specified range of tracks

For type 1 and type 2 calls, the information requested is returned directly in the csParam field. For type 3 calls, the user must allocate a data buffer for the data returned by the driver.

The following additional parameters are needed for type 3 calls:

csParam+2 (long)	address of data buffer
csParam+6 (word)	length of data buffer
csParam+8 (byte)	starting track number in BCD

The driver calculates the number of tracks from the length of the data buffer. If you allocate too little or too much space, undefined errors may result.

Output:

For Type 1 calls:

csParam+0 (byte)	First track number in BCD
csParam+1 (byte)	Last track number in BCD

For Type 2 calls:

csParam+0 (byte)	lead out (end of disc) address, minutes, in BCD
csParam+1 (byte)	lead out (end of disc) address, seconds, in BCD
csParam+2 (byte)	lead out (end of disc) address, frames, in BCD

For Type 3 calls:
Four bytes of data will be returned in the buffer specified, for each track, in the following format:

Byte	Content
0	Control field of specified track (see control field bit description)
1	track start, absolute minutes, in BCD
2	track start, absolute seconds, in BCD
3	track start, absolute frames, in BCD

Return codes:	NoErr	No error
	badUnitErr	Bad reference number
	unitEmptyErr	Bad reference number
	notOpenErr	Driver is not open
	paramErr	Bad parameter error
	controlErr	Driver can't respond to this control call

ReadQ (101)

Description: The control call `ReadQ` returns the information stored in the current Q subcode channel in `csParam`. The Q channel contains control information, track number, and location in minutes, seconds, and frames.

Input: none

Output:	csParam +0	Control field (described earlier)
	csParam +1	Track number in BCD (range 01 to 99)
	csParam +2	Index (X)
	csParam +3	address in minutes from start of track in BCD
	csParam +4	address in seconds from start of track in BCD
	csParam +5	address in frames from start of track in BCD
	csParam +6	address in absolute minutes from start of disc in BCD
	csParam +7	address in absolute seconds from start of disc in BCD
	csParam +8	address in absolute frames from start of disc in BCD

Return codes:	NoErr	No error
	badUnitErr	Bad reference number
	unitEmptyErr	Bad reference number
	notOpenErr	Driver is not open
	paramErr	Bad parameter error
	controlErr	Driver can't respond to this control call

ReadHead (102)

Description: The control call `ReadHead` returns the absolute minute-second-frame address and play mode information about a specific logical block. If a requested block is in a CD-Audio section of the disc, a `paramErr` will be returned. An application program can find out what mode of a block is easier with this command than with the `Read Extended` command.

Input:	<code>csParam +0</code> (long)	Logical block number
Output:	<code>csParam +0</code> (byte)	Absolute time in minutes in BCD
	<code>csParam +1</code> (byte)	Absolute time in seconds in BCD
	<code>csParam +2</code> (byte)	Absolute time in frames in BCD
	<code>csParam +3</code> (byte)	Play mode
Return codes:	<code>NoErr</code>	Current block is a data block
	<code>badUnitErr</code>	Bad reference number
	<code>unitEmptyErr</code>	Bad reference number
	<code>notOpenErr</code>	Driver is not open
	<code>paramErr</code>	Current block is a audio block
	<code>controlErr</code>	Driver can't respond to this control call

ATrkSearch (103)

Description: The control call `ATrkSearch` positions the optical pickup at the specified audio address. A `paramErr` error is returned if the address is not in a CD-Audio area of the disc (either out of bounds or a data track). The type of address is specified in `csParam+0` with the additional parameters in `csParam+2` through 9.

Input:

<code>csParam +0</code> (word)	0x0000 Type 0: Search address in logical block format
<code>csParam +2</code> (long)	Logical block address from MSB in byte 2 to LSB in byte 5
<code>csParam +6</code> (word)	Play flag 0 = search and pause at this address 1 = search and play according to play mode
<code>csParam +8,9</code> (word)	8 = 00, 9 = Play mode (see play mode description)
<code>csParam +0</code> (word)	0x0001 Type 1: Search address in absolute minute-second-frame format
<code>csParam +2</code> (byte)	00
<code>csParam +3</code> (byte)	Absolute minutes in BCD
<code>csParam +4</code> (byte)	Absolute seconds in BCD
<code>csParam +5</code> (byte)	Absolute frames in BCD
<code>csParam +6</code> (word)	Play flag (described earlier) 0 = search and pause at address 1 = search and play according to play mode
<code>csParam +8,9</code> (word)	8 = 00, 9 = Play mode (see play mode description)
<code>csParam +0</code> (word)	0x0002 Type 2: Search address in track number format in BCD
<code>csParam +2</code> (byte)	00
<code>csParam +3</code> (byte)	00
<code>csParam +4</code> (byte)	00
<code>csParam +5</code> (byte)	Track number in BCD
<code>csParam +6</code> (word)	Play flag (described earlier) 0 = search and pause at address 1 = search and play according to play mode
<code>csParam +8,9</code> (word)	8 = 00, 9 = Play mode (see play mode description)

Output: none

Return codes:	NoErr	No error
	badUnitErr	Bad reference number
	unitEmptyErr	Bad reference number
	notOpenErr	Driver is not open
	paramErr	Bad parameter error
	controlErr	Driver can't respond to this control call

APlay (104)

Description: The control call `APlay` positions the optical pickup at the specified audio address and begins playback in the specified play mode when the address is found. A `ParamErr` is returned if the specified address is a data track.

If you haven't issued an `AStop` before issuing an `APlay`, the AppleCD SC plays from the beginning of the specified track to the end of the disc and stops.

Input:	<code>csParam +0</code> (word)	0x0000 Type 0: Play address in logical block format
	<code>csParam +2</code> (long)	Logical block address from MSB in byte 2 to LSB in byte 5
	<code>csParam +6</code> (word)	Stop address flag 0 = start play at this address 1 = stop play at this address
	<code>csParam +8,9</code> (word)	8 = 00, 9 = Play mode (see play mode description)
	<code>csParam +0</code> (word)	0x0001 Type 1: Play address in absolute minute-second-frame format
	<code>csParam +2</code> (byte)	00
	<code>csParam +3</code> (byte)	Absolute minutes in BCD
	<code>csParam +4</code> (byte)	Absolute seconds in BCD
	<code>csParam +5</code> (byte)	Absolute frames in BCD
	<code>csParam +6</code> (word)	Stop address flag 0 = start play at this address 1 = stop play at this address
	<code>csParam +8,9</code> (word)	8 = 00, 9 = Play mode (see play mode description)

csParam +0 (word)	0x0002	Type 2: Play address in track number format in BCD
csParam +2 (byte)	00	
csParam +3 (byte)	00	
csParam +4 (byte)	00	
csParam +5 (byte)		Track number in BCD
csParam +6 (word)		Stop address flag
	0	= start play at this address
	1	= stop play at this address
csParam +8,9 (word)	8 = 00, 9 =	Play mode (see play mode description)

Output: none

Return codes:

NoErr	No error
badUnitErr	Bad reference number
unitEmptyErr	Bad reference number
notOpenErr	Driver is not open
paramErr	Bad parameter error
controlErr	Driver can't respond to this control call

APause (105)

Description: The control call `APause` starts or stops the current `APlay` operation. This call is undefined if a prior `APlay` or `ATrkSearch` has not been called.

Input:

csParam +0 (long)	
0x00000001	enter hold state and mute audio
0x00000000	release pause and resume play at next block

Output: none

Return codes:

NoErr	No error
badUnitErr	Bad reference number
unitEmptyErr	Bad reference number
notOpenErr	Driver is not open
paramErr	Bad parameter error
controlErr	Driver can't respond to this control call

AStop (106)

Description: The control call `AStop` specifies the audio address at which `APlay` operation will stop. The `AStop` call actually plays the track specified and stops at the end of that specified track.

Input:

<code>csParam +0</code> (word)	<code>0x0000</code> Type 0: Stop address logical block format
<code>csParam +2</code> (long)	Logical block address from MSB in byte 2 to LSB in byte 5
<code>csParam +0</code> (word)	<code>0x0001</code> Type 1: Stop address in absolute minutes-seconds-frames format
<code>csParam +2</code> (byte)	00
<code>csParam +3</code> (byte)	Absolute minutes in BCD
<code>csParam +4</code> (byte)	Absolute seconds in BCD
<code>csParam +5</code> (byte)	Absolute frames in BCD
<code>csParam +0</code> (word)	<code>0x0002</code> Type 2: Stop address in track number format
<code>csParam +2</code> (byte)	00
<code>csParam +3</code> (byte)	00
<code>csParam +4</code> (byte)	00
<code>csParam +5</code> (byte)	Track number in BCD

Output: none

Return codes:

<code>NoErr</code>	No error
<code>badUnitErr</code>	Bad reference number
<code>unitEmptyErr</code>	Bad reference number
<code>notOpenErr</code>	Driver is not open
<code>paramErr</code>	Bad parameter error
<code>controlErr</code>	Driver can't respond to this control call

AStatus (107)

Description: The control call `AStatus` returns the status of the current audio track. If the AppleCD SC drive is not in the ready condition, an `ioErr` error will be returned. (The application may then call `GetSense` to request sense bytes describing the error condition.) If `AStatus` is called while the AppleCD SC is reading a data block, `ioErr` will be returned. The call returns information about the type of play in progress, the type of track, and the current absolute location on the disc.

Input: none

Output:

<code>csParam +0</code> (byte)	Audio status: 0: audio play in progress 1: audio pause in operation 2: audio muting on 3: audio play operation completed 4: error occurred during audio play 5: not currently playing
<code>csParam +1</code> (byte)	Play mode (see play mode description)
<code>csParam +2</code> (byte)	Control field (see control field description)
<code>csParam +3</code> (byte)	Absolute minutes in BCD
<code>csParam +4</code> (byte)	Absolute seconds in BCD
<code>csParam +5</code> (byte)	Absolute frames in BCD

Return codes:

<code>NoErr</code>	No error
<code>badUnitErr</code>	Bad reference number
<code>unitEmptyErr</code>	Bad reference number
<code>notOpenErr</code>	Driver is not open
<code>paramErr</code>	Bad parameter error
<code>controlErr</code>	Driver can't respond to this control call

AScan (108)

Description: This control call performs a fast-forward or fast-reverse scan operation on the AppleCD SC starting from the specified address.

Input:

csParam +0 (word)	0x0000 Type 0: Scan address logical block format
csParam +2 (long)	Logical block address from MSB in byte 2 to LSB in byte 5
csParam +6 (word)	Direction of scan 0x0000: fast forward 0x0001: fast reverse
csParam +0 (word)	0x0001 Type 1: Scan address in absolute minutes-seconds-frames format
csParam +2 (byte)	00
csParam +3 (byte)	Absolute minutes in BCD
csParam +4 (byte)	Absolute seconds in BCD
csParam +5 (byte)	Absolute frames in BCD
csParam +6 (word)	Direction of scan 0x0000: fast forward 0x0001: fast reverse
csParam +0 (word)	0x0002 Type 2: search address track number format
csParam +2 (byte)	00
csParam +3 (byte)	00
csParam +4 (byte)	00
csParam +5 (byte)	Track number in BCD
csParam +6 (word)	Direction of scan 0x0000: fast forward 0x0001: fast reverse

Output: none

Return codes:

NoErr	No error
badUnitErr	Bad reference number
unitEmptyErr	Bad reference number
notOpenErr	Driver is not open
paramErr	Bad parameter error
controlErr	Driver can't respond to this control call

Macintosh CD-ROM device driver status calls

The Apple CD-ROM device driver status calls are described in this section. Many of these calls are standard Apple SCSI device calls; however, the data structures and values provided in the output of these call descriptions are specific to the AppleCD SC.

DiskStatus (8)

Description: The `DiskStatus` call is used by the system to learn various attributes of the currently mounted disc.

Input: none

Output:

<code>csParam +0</code>	<code>track = 0</code>
<code>csParam +2</code>	<code>WriteProtect = 128</code>
<code>csParam +3</code>	<code>diskInPlace 1=yes 0=no</code>
<code>csParam +4</code>	<code>installed = 1</code>
<code>csParam +5</code>	<code>sides = 1</code>
<code>csParam +6</code>	<code>qLink = NIL</code>
<code>csParam +10</code>	<code>qType = 0</code>
<code>csParam +12</code>	<code>dQDrive = drive number</code>
<code>csParam +14</code>	<code>dQRefNum = drive reference number</code>
<code>csParam +16</code>	<code>dQFSID = 1</code>
<code>csParam +18</code>	<code>twoSideFmt = 0</code>
<code>csParam +19</code>	<code>needsFlush = 0</code>
<code>csParam +20</code>	<code>diskErrs = 0</code>

Return codes:

<code>NoErr</code>	No error
<code>badUnitErr</code>	Bad reference number
<code>unitEmptyErr</code>	Bad reference number
<code>notOpenErr</code>	Driver is not open

WhoIsThere (97)

Description: WhoIsThere lets an application know which SCSI IDs are AppleCD SC drives.

Input: none

Output: csParam +0 00
csParam +1 (byte)
1 byte with bits 7 through 0 corresponding to SCSI IDs 7 through 0. If bit 1 equals 1, the AppleCD SC has a SCSI ID of 1 assigned to it. Bits are ordered 7, 6, 5, 4, 3, 2, 1, 0, with the MSB leftmost in the byte. For example, if two CD SC drives had SCSI ID numbers 7 and 1, the IDs would be represented by the bit values 10000010 in the byte.

Return codes:

NoErr	No error
badUnitErr	Bad reference number
unitEmptyErr	Bad reference number
notOpenErr	Driver is not open

GetSize (98)

Description: GetSize returns the currently assigned logical block size of the AppleCD SC. The default block size for the Macintosh is 512-byte blocks. Block size can be set with the varBlk control call described earlier. Most CD-ROM discs have a block size of 2048 bytes.

Input: none

Output: csParam +0 (word)
Block size

Return codes:

NoErr	No error
badUnitErr	Bad reference number
unitEmptyErr	Bad reference number
notOpenErr	Driver is not open

GetSense (99)

Description: GetSense returns the current sense bytes to the calling application. The calling application should make this call after receiving the `ioErr` return code. See Appendix A for a description of the sense codes.

Input: none

Output: csParam +0 18 bytes of sense data

Return codes: NoErr No error
 badUnitErr Bad reference number
 unitEmptyErr Bad reference number
 notOpenErr Driver is not open

Apple II AppleCD SC SmartPort control call descriptions

This section provides a brief overview of the Apple II SmartPort interface and descriptions of the AppleCD SC control calls supported by the SmartPort. Note that the parameter list for the AppleCD SC control calls is the same as described in the firmware command definitions section of the *Apple II SCSI Card Technical Reference Manual* unless otherwise defined in the control call descriptions.

- ◆ *Note:* The data structures for the control calls described in this chapter are specific to the Apple II ProDOS SmartPort environment, which means they are valid on an Apple IIe and Apple IIGS running ProDOS. However, the specific information needed for making control calls in the Apple IIGS GS/OS environment is not provided in this chapter. See the *GS/OS Reference* for a complete description of the AppleCD SC device driver interface in the GS/OS environment.

The SmartPort interface

In order to access and control a SCSI device such as the AppleCD SC, the host computer must be able to send and receive SCSI commands from either the operating system or the active application program. Rather than force each application or operating system to provide all of the functions necessary to send and receive SCSI commands for each device, Apple has created a command interpreter in the Apple II SCSI card's firmware. This interpreter is the SmartPort I/O interface. The SCSI card receives SmartPort control calls from your application and ensures that the requested command is sent to the device in the appropriate format.

Requests from the SmartPort are much like routines, in that the command interpreter takes a simple input, such as Status, and uses several lower-level SCSI commands, such as Mode Sense and Request Sense, to accomplish the operation requested.

Even with the SmartPort command interpreter, you still need to provide the parameter data (depending on the command) necessary to accomplish the operation requested. For instance, the interpreter would select the correct commands to accomplish a `ReadBlock` call, but would not be able to return any data read from the target device unless you had specified the block location of the data on the target in the original SmartPort call.

A SmartPort call includes a *command byte*, a *pointer*, and a *parameter list*.

A **command byte** is a hexadecimal byte that indicates to the interpreter the operation being requested. Each operation has a unique command byte. For a SmartPort control call, the command byte is always \$04. SCSI commands also have a control call code, but this code is not directly related to the command byte sent to the interpreter in a SmartPort call. *Do not confuse SCSI command byte numbers with ProDOS/SmartPort control call codes.* The SmartPort control call codes are actually the first bytes in the control call parameter list, which is described later.

A **pointer** is a data field that contains the address in RAM of the SmartPort AppleCD SC control call parameter list.

A **parameter list** is a set of related data fields loaded into RAM at a specific address. Parameter lists can contain special codes, pointers to RAM locations loaded with outgoing data blocks, control variables, or any other information the command requires to be executed properly.

Basically, you build the parameter list in Apple II RAM, load the RAM address of your list into the pointer field, make the appropriate call to the command interpreter via the SmartPort, and wait for the call to finish.

Once a call is issued, the Apple II turns control over to the interpreter until the requested operation is done. When the interpreter completes the operation, it returns control to the program issuing the original call. Program execution resumes at the address immediately following the parameter list pointer for the call. Table 7-2 shows the SmartPort control call codes and associated call names.

△ **Important** To fully understand and implement the AppleCD SC control calls through the Apple II SmartPort interface described in this section, you need to refer to the description of the Apple II SCSI Card SmartPort firmware provided in the *Apple II SCSI Card Technical Reference Manual*. △

■ **Table 7-2** AppleCD SC SmartPort control call codes

SmartPort code	Definition
\$04	Eject
\$05	TestUnitReady
\$06	RequestSense
\$08	ModeSelect
\$09	ModeSense
\$0D	ReadCapacity
\$0F	Inquiry
\$13	SetBlockSize
\$14	SetTimeout
\$16	ExtendedSeek
\$18	StartUnit
\$19	StopUnit
\$1A	PreventRemoval
\$1B	AllowRemoval
\$1C	Verify
\$1D	RezeroUnit
\$1E	Patch1Call
\$1F	SetNewSDAT
\$20	AudioSearch
\$21	AudioPlay
\$22	AudioPause
\$23	AudioStop
\$24	AudioStatus
\$25	AudioScan
\$26	Eject
\$27	ReadTOC
\$28	ReadQSubcode
\$29	ReadHeader

AppleCD SC SmartPort control calls

The AppleCD SC SmartPort control calls are described in detail below. Each definition contains the following information.

Call name (control code)

Description: A brief statement of the purpose or function of the control call

Byte: The byte number(s) of the parameter in the SmartPort parameter list

Definition: Description of the parameters in the parameter list

The SmartPort control calls described in this section are given in the Apple IIGS extended format. The extended format requires a 24-bit (or greater) address field. However, the standard format used for programming the Apple IIe requires a 16-bit address field. For example, in the `Eject` call parameter list, which follows, the extended format requires that (4 bytes) bytes 2 through 5 are reserved and the control code is in byte 6. To convert this call into the standard format, change the parameter list so that (2 bytes) bytes 2 and 3 are reserved and the control code is in byte 4. Make this same conversion for any of the commands described in this section when programming specifically for an Apple IIe.

Eject (\$04)

Description: The SmartPort `Eject` control code executes the AppleCD SC SCSI `Eject` command, which ejects the CD-ROM caddy. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	Unit number of the target AppleCD SC
2-5	Reserved (\$00)
6	Control code (\$04)

TestUnitReady (\$05)

Description: The `TestUnitReady` SmartPort control code executes AppleCD SC SCSI `Test Unit Ready` command, which determines if the AppleCD SC drive is turned on and ready for operation. Refer to the ANSI X3T9.2/82-2 Rev. 17B specification for details regarding this standard group 0 command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	Unit number of the target AppleCD SC
2-5	Reserved (\$00)
6	Control code (\$05)

RequestSense (\$06)

Description: The `RequestSense` SmartPort control code executes the AppleCD SC SCSI `Request Sense` command. The returned `Request Sense` data follows the ANSI X3T9.2/82-2 Rev. 17B specification. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	Unit number of the target AppleCD SC
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address where you want the sense data returned. See Appendix A for the additional sense codes (byte 12).
6	Control code (\$06)

If your application requires additional sense data from the `Request Sense` command, additional sense codes are returned in byte 12 of the extended sense data format. The sense codes are defined in Appendix A. The standard SCSI sense keys (bits 0 through 3 of byte 2) that are associated with the additional sense code are provided in the column labeled "Related sense keys." If the target does not have any additional sense data to send, this field is set to \$00. The format of the sense data returned by the `RequestSense` call is shown in Figure 7-3.

■ **Figure 7-3** Request Sense return sense data format

		Bit Number							
		7	6	5	4	3	2	1	0
Byte Number	0	valid	error class \$07				error code \$00		
	1	segment number \$00							
	2	fm	eom	lll	rsvd	sense key			
	3	information byte (msb)							
	4	information byte							
	5	information byte							
	6	information byte (lsb)							
	7	additional sense length (n)							
	8	additional sense information							
	9	additional sense information							
	10	additional sense information							
	11	additional sense information							
	12	additional sense code							
	13	reserved							
	14	fru failed							
	15	fpv	c/d	v	v	bpv	bit pointer		

ModeSelect (\$08)

Description: The ModeSelect SmartPort control code executes the AppleCD SC SCSI Mode Select command. The Mode Select command passes a set of parameters to the AppleCD SC. The parameter information specifies or changes medium, logical unit, target, or device operating parameters to the AppleCD SC. The information is passed to the target in the Mode Select parameter list you build in Apple II memory. The parameters for the AppleCD SC that differ from the ANSI specification are described later. The SmartPort parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$04)
1	Unit number
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address that contains the Mode Select parameters you want to send to the AppleCD SC.
6	Control code (\$08)
7	Transfer byte count

The following values apply to the AppleCD SC Mode Select parameter list.

Mode Select Header: The medium type, byte 1, is set to a value of \$00 for the AppleCD SC. The block descriptor length, byte 3, specifies the length, in bytes, of the block descriptor to be passed to the target. Because the AppleCD SC can have at most only one block descriptor, a value of \$08 indicates that there is a block descriptor and a value of \$00 indicates that no block descriptor shall be passed in the parameter list; *this is not an error condition*. The format of the Mode Select header is shown in Figure 7-4.

■ **Figure 7-4** Mode Select header data format

	Bit number							
	7	6	5	4	3	2	1	0
Byte number	0	Reserved						
	1	Medium type = \$00						
	2	Reserved						
	3	Block descriptor length = (\$00 or \$08)						

Block descriptor: The block descriptor defines the logical block length on the medium. The block descriptor is 8 bytes long.

A value of \$00 in the number of blocks fields (bytes 1–3) specifies the total logical blocks of the disc currently in the AppleCD SC.

The block length field (bytes 5–7) contains the logical block size in bytes to be used for the blocks configured according to the parameters in this block descriptor. Block lengths of 256 (\$100), 512 (\$200), 1024 (\$400), 2048 (\$800), 2052 (\$804), 2336 (\$920), 2340 (\$924), or 2352 (\$930) bytes are supported by the AppleCD SC. The default block length value for the AppleCD SC is 2048 bytes. The format for the Mode Select block descriptor is shown in Figure 7-5.

■ **Figure 7-5** Mode Select block descriptor format

	Bit number							
	7	6	5	4	3	2	1	0
Byte number	0	Reserved						
	1	Number of blocks (MSB) = \$00						
	2	Number of blocks = \$00						
	3	Number of blocks (LSB) = \$00						
	4	Reserved						
	5	Block length (MSB)						
	6	Block length						
	7	Block length (LSB)						

- ◆ *Note:* If you set the block length to 2336, 2340, or 2352 and the LECC is enabled, the AppleCD SC will generate an illegal request Check Condition for both mode 1 and mode 2 data blocks. You should turn layered error correction off if you need to access mode 2 data blocks of sizes 2336, 2340, or 2352. Accessing mode 2 data blocks of lengths other than 2336, 2340, or 2352 will return a Check Condition. (To turn off layered error correction, see the DCR description in the error recovery page definition later in this Mode Select command description.)

Page descriptors: Each page descriptor contains a 2-byte header defining the page code and the length of the page. The pages are separated into subblocks containing a list of related flags and values. The page code field contains the Apple Common Command Set (CCS) code indicating the format and contents of page data for the specific page descriptor. The format of the Mode Select page descriptor is shown in Figure 7-6.

■ **Figure 7-6** Mode Select page descriptor format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Rsvd	Rsvd	Page code					
	1	Page length in bytes							
	2 to n	Refer to the definition of the page selected							

◆ *Note:* Bits 7 and 6 in byte 0 of each page descriptor are reserved.

Page codes for the AppleCD SC Mode Select and Mode Sense commands are shown in Table 7-3.

■ **Table 7-3** Apple Common Command Set page codes

Code	Page data type
\$00	Unit attention
\$01	Error recovery
\$02	Disconnect/reconnect control
\$20	Drive parameters
\$21–\$3E	Vendor specific
\$3F	Return all valid pages (Mode Sense command only)

The format of the Mode Select unit attention page, page code \$00, is shown in Figure 7-7 and described in the text that follows.

■ **Figure 7-7** Mode Select unit attention page format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Rsvd	Rsvd	Page code =\$00					
	1	Page length in bytes							
	2	VS	VS	VS	UATT	Reserved			
	3	Reserved							

Page code: The page code field, bits 0–5 of byte 0 in the unit attention page, is set to \$00 for the unit attention page.

Page length: The page length field, byte 1 of the unit attention page, is set to 2, which indicates the length of the unit attention page in bytes.

Vendor specific (vs): The vendor-specific field, bits 5–7 of byte 2 of the unit attention page, contains any vendor-specific flag supported by the AppleCD SC that is not a part of the Apple Common Command Set.

UATT: The UATT field, bit 4 of byte 2 of the unit attention page, contains the unit attention flag.

When the UATT flag is set to 1, it indicates that the AppleCD SC will accept a `Read` command with the logical block address set to 0, an `Inquiry` command, or a `Request Sense` command following a hard reset. Any other command causes the AppleCD SC to return the Check Condition status and set the Request Sense sense key to Unit Attention. All other reset conditions, a power on, a `Mode Select` parameter change, or a medium change return the Check Condition status and set the `RequestSense` sense key to Unit Attention.

When the UATT flag is set to 0, the AppleCD SC will return the Check Condition status and set the sense key to Unit Attention following all reset conditions, `Mode Select` parameter change, and medium change.

Reserved: All reserved bit or byte fields are set to \$00.

Error recovery page: The error recovery page is used to control the execution of error recovery operations on the AppleCD SC. The format of the error recovery page is shown in Figure 7-8. The fields in the error recovery page are defined below.

■ **Figure 7-8** Mode Select error recovery page format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Rsvd	Rsvd	Page code = \$01					
	1	Page length in bytes							
	2	Rsvd	Rsvd	TB	RC	ECC	PER	DTE	DCR
	3	Retry count							
	4	Reserved							
	5	Reserved							
	6	Reserved							
	7	Recovery time limit = \$00							

Page code: The page code field, bits 0–5 of byte 0 in the error recovery page, is set to \$01 for the error recovery page.

Page length: The page length field, byte 1 of the error recovery page, is set to 6, the length of the error recovery page in bytes.

The descriptions of the DCR, DTE, PER, EEC, RC, and TB error recovery bits that follow are contained in byte 2 of the error recovery page

DCR: This field, bit 0, contains the disable error correction flag. When set to 1, this flag indicates that the AppleCD SC is to transfer data without layered error correction (LECC). When set to 0, this flag indicates that the AppleCD SC is to enable layered error correction to all data transferred. The default value of the DCR flag is 0.

DTE: This is the disable transfer on error field, bit 1. When DTE and PER are both set to 1 and an error is detected, the AppleCD SC terminates the currently executing command with the Check Condition status and sets the appropriate sense key for all errors encountered. Additionally, the data of the block in error, which is the first erring block encountered, may or may not be transferred to the initiator depending upon the setting of the TB bit. When the DTE bit is set to 0, the AppleCD SC continues data transfer after encountering recovered errors up to the error recovery flag limits. Any erring block that would be posted, which is the last recovered block encountered, is not posted until the Transfer Length is exhausted. The default value of the DTE flag is 0.

- △ **Important** If PER is set to 0 and DTE is set to 1, the AppleCD SC will terminate the Mode Select command with the Check Condition status and set the sense key to illegal request. See Table 7-4 for valid bit settings. △

PER: A PER (post error) bit 2 set to 1 indicates that the AppleCD SC shall report Check Condition status for recovered errors, with the appropriate sense key. The Check Condition shall happen during the data transfer depending either on the DTE bit value, or if an unrecoverable error occurred. If multiple errors occur, the Request Sense data shall report the block address of either the last block on which the recovered error occurred or of the first unrecoverable error.

A PER bit set to 0 indicates that the AppleCD SC shall not create the Check Condition status for errors recovered within the limits established by the other error recovery flags. Recovery procedures exceeding the limits established by the other error recovery flags shall be posted accordingly by the AppleCD SC. The transfer of data may terminate prior to exhausting the transfer length, depending on the error and the state of the other error recovery flags. The default value of the PER flag is 0.

EEC: An EEC (enable early correction) bit 3 set to 1 indicates that the AppleCD SC shall enable the use of the most expedient form of error recovery, such as error correction, before applying retries. Seek or positioning retries and the recovery procedure retries of the message system are not affected by the value of this bit. If EEC and DCR are both set to 1, it is an invalid request, for which the AppleCD SC shall create the Check Condition status with an Illegal Request sense key.

An EEC bit set to 0 indicates that the AppleCD SC shall exhaust the defined retry limit prior to enabling error correction. If the DCR bit is set to 1, the defined retry limit only is to be performed. The default value of this flag is 0.

RC: An RC (read continuous) bit 4 set to 1 requests the AppleCD SC to transfer the entire requested length of data without adding delays that would increase or ensure data integrity (that is, delays caused by the error recovery schemes). This setting implies that the AppleCD SC may send data that may be erroneous or fabricated in order to maintain a continuous flow of data and avoid delays.

The AppleCD SC shall assign priority to this bit over conflicting error control bits (EEC, DCR, DTE, PER) within this byte.

An RC bit set to 0 indicates that error recovery operations that cause reasonable delays are acceptable during the data transfer. Data shall not be fabricated. The RC field is reserved for the AppleCD SC and has a default value of 0.

- ◆ *Note:* The AppleCD SC has the additional requirement that delays can occur only between transfer of two data blocks, but not within a data block.

TB: A TB (transfer block) bit 5 set to 1 indicates that the data in the failing data block (recovered or unrecoverable) shall be transferred to the initiator. A TB bit set to zero indicates that the data in the failing data block (recovered or unrecoverable) shall not be transferred to the initiator. The default value for the TB bit is 0 for the AppleCD SC.

In both cases, but particularly when TB is zero, the block address reported in the Request Sense data shall be that of the erring block, not of the preceding block.

Table 7-4 shows all of the valid byte values and bit settings for the AppleCD SC error recovery page. The AppleCD SC error recovery procedures that correspond to the error recovery byte values are described later.

■ **Table 7-4** Valid error recovery page byte values for the AppleCD SC modes of operation

Byte values	Bit settings							
	Rsvd	Rsvd	TB	Rsvd	EEC	PER	DTE	DCR
\$00	0	0	0	0	0	0	0	0
\$01	0	0	0	0	0	0	0	1
\$04	0	0	0	0	0	1	0	0
\$05	0	0	0	0	0	1	0	1
\$06	0	0	0	0	0	1	1	0
\$07	0	0	0	0	0	1	1	1
\$20	0	0	1	0	0	0	0	0
\$21	0	0	1	0	0	0	0	1
\$26	0	0	1	0	0	1	1	0
\$27	0	0	1	0	0	1	1	1

Retry count: Byte 3 of the error recovery parameter page contains the retry limit field. The limit is the maximum number of times the AppleCD SC executes its retry algorithm before terminating the currently executing command with the Check Condition status and setting the appropriate sense key. If DCR is set to 1, retry count is ignored during command execution. It is possible to set this field to a nonzero number even if DCR is set to 1, since DCR may be set to 0 with a subsequent Mode Select command. The default value of this field is \$00.

Recovery time limit: Byte 7 of the error recovery parameter page contains the amount of time that can elapse before the AppleCD SC ceases error recovery operations. A value of 0 indicates that there is no time limit. The value for the AppleCD SC recovery time limit is 0.

Error recovery procedures for the AppleCD SC mode 1 data block are described in Table 7-5.

■ **Table 7-5** CD-ROM error recovery procedures

Error recovery byte value	Description
\$00	This is the default value of the error recovery byte. LECC is on. Only LECC uncorrectable data errors are reported. If an uncorrectable data error occurs, data transfer is terminated at the beginning of the first encountered uncorrectable block with a check condition status. The sense key is set to medium error and the additional sense code is set to LECC uncorrectable data error. The information bytes returned by the <code>Request Sense</code> command are set to the address of the first encountered uncorrectable block address.
\$04	LECC is on. LECC recovered data block errors are reported. If a LECC recovered data error occurs, the data transfer is not terminated until all requested data blocks (including the LECC recovered data block) are transferred to the initiator. After the data transfer is completed, a Check Condition status is reported. The sense key is set to recovered error and the additional sense code is set to LECC recovered data error. The information bytes are set to the address of the last block for which a LECC recovered data error was detected. If a LECC uncorrectable data error occurs, data transfer is not terminated until all requested data blocks (including the LECC uncorrectable data block) are transferred to the initiator. After the data transfer is completed, a Check Condition status is reported. The sense key is set to medium error and the additional sense code is set to LECC uncorrectable error. The information bytes are set to the address of the last block for which a LECC uncorrectable error was detected.
\$06	LECC is on. LECC recovered data errors are reported. If a LECC recovered data error occurs, data transfer is terminated at the beginning of the first encountered LECC recovered data block with a Check Condition status. The sense key is set to recovered error and the additional sense code is set to LECC recovered data error. The information bytes are set to the address of the first encountered LECC recovered error block. If a LECC uncorrectable data error occurs, data transfer is terminated at the beginning of the first encountered LECC uncorrectable data block with a Check Condition status. The sense key is set to medium error and the additional sense code is set to LECC uncorrectable error. The information bytes are set to the address of the first encountered LECC uncorrectable block.

■ **Table 7-5** CD-ROM error recovery procedures (continued)

Error recovery byte value	Description
\$26	LECC is on. LECC recovered data errors are reported. This setting has the same function as the setting of \$06 except that if a LECC recovered or a LECC uncorrectable data error occurs, data transfer is terminated at the end of the first encountered LECC recovered or LECC uncorrectable data block with a Check Condition status.
\$20	LECC is on. Only LECC uncorrected data errors are reported. If a LECC uncorrectable data block error occurs, the data transfer is not terminated until all requested data blocks (including the LECC uncorrectable data block) are transferred to the initiator. After the data transfer has completed, a Check Condition status is reported. The sense key is set to medium error and the additional sense code is set to LECC uncorrectable error. The information bytes are set to the address of the last block for which a LECC uncorrectable data error was detected. The error recovery procedures that follow apply to both the AppleCD SC mode 1 and mode 2 data block.
\$01	LECC is not used. Only CIRC unrecovered data errors are reported. If a CIRC unrecovered data error occurs, data transfer is terminated at the beginning of the first encountered CIRC unrecovered data block with a Check Condition status. The sense key is set to medium error and the additional sense code is set to CIRC unrecovered data error. The information bytes are set to the address of the first encountered CIRC unrecovered block address.
\$05	LECC is not used. CIRC recovered data block errors are reported. If a CIRC recovered data block error occurs, the data transfer is not terminated until all requested data blocks (including the CIRC recovered data block) are transferred to the initiator. After the data transfer is completed, a Check Condition status is reported. The sense key is set to recovered error and the additional sense code is set to CIRC recovered data error. The information bytes are set to the address of the last block for which a CIRC recovered data error was detected. If a CIRC unrecoverable data error occurs, data transfer is not terminated until all requested data blocks (including the CIRC unrecovered data block) are transferred to the initiator. The sense key is set to medium error and the additional sense code is set to CIRC unrecovered data error. The information bytes are set to the address of the last-encountered CIRC unrecovered block.

■ **Table 7-5** CD-ROM error recovery procedures (continued)

Error recovery byte value	Description
\$07	LECC is not used. CIRC recovered data errors are reported. If a LECC recovered data error occurs, data transfer is terminated at the beginning of the first-encountered CIRC recovered data block with a Check Condition status. The sense key is set to recovered error and the additional sense code is set to CIRC recovered data error. The information bytes are set to the address of the first-encountered CIRC recovered error block. If a CIRC unrecovered data error occurs, data transfer is terminated at the beginning of the first-encountered CIRC unrecovered data block with a Check Condition status. The sense key is set to medium error and the additional sense code is set to CIRC unrecovered error. The information bytes are set to the address of the first-encountered CIRC unrecovered block.
\$27	LECC is not used. CIRC recovered data errors are reported. This setting has the same function as the setting of \$07, except that if a CIRC recovered or CIRC unrecovered data error occurs, data transfer is terminated at the end of the first-encountered CIRC recovered or CIRC unrecovered data block with a Check Condition status.
\$21H	LECC is not used. Only CIRC unrecovered data errors are reported. If a CIRC unrecovered data block error occurs, the data transfer is not terminated until all requested data blocks (including the CIRC unrecovered data block) are transferred to the initiator. After the data transfer has completed, a Check Condition status is reported. The sense key is set to medium error and the additional sense code is set to CIRC unrecovered error. The information bytes are set to the address of the last block for which a CIRC unrecovered data error was detected.

Disconnect/reconnect control page: The disconnect/reconnect control page is used to control the operation of the AppleCD SC following a reset condition. The format for the disconnect/reconnect control page is shown in Figure 7-9. The fields in the disconnect/reconnect control page are defined below.

■ **Figure 7-9** Mode Select disconnect/reconnect page format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Rsvd	Rsvd	Page code = \$02					
	1	Page length in bytes							
	2	Buffer full ratio							
	3	Reserved							
	4	Bus inactivity limit (MSB)							
	5	Bus inactivity limit (LSB)							
	6	Disconnect limit (MSB)							
	7	Disconnect limit (LSB)							
	8	Connect limit (MSB)							
	9	Connect limit (LSB)							
	10	Reserved							
	11	Reserved							

Page code: The page code field, bits 0–5 of byte 0, is set to \$02 for the unit attention page.

Page length: The page length field, byte 1, is set to a value of 10, the length of the disconnect/reconnect control page in bytes.

Buffer full ratio: The buffer full ratio field, byte 2, contains a variable that determines how full the AppleCD SC's buffer will be before reconnect occurs for output during Read command execution.

All you need to do to set the reconnect point is pass the right value in buffer full; the sizing calculations are done by the AppleCD SC. The relationships between the values of the buffer full ratio and the number of blocks are specified in Table 7-6.

- ◆ *Note:* Remember to load your buffer full variable into the page descriptor as a hexadecimal value; the AppleCD SC does its calculations assuming that the value loaded into buffer full is in hexadecimal.

■ **Table 7-6** Relationships between buffer full ratio and number of blocks

Value of buffer full ratio	Number of 2048-byte blocks	Number of 2336/2340-byte blocks	Value of buffer full ratio	Number of 2048-byte blocks	Number of 2336/2340-byte blocks
\$00	32	28	\$81-\$88	17	15
\$01-\$08	1	1	\$89-\$90	18	16
\$09-\$10	2	2	\$91-\$98	19	17
\$11-\$18	3	3	\$99-\$A0	20	18
\$19-\$20	4	4	\$A1-\$A8	21	19
\$21-\$28	5	5	\$A9-\$B0	22	20
\$29-\$30	6	6	\$B1-\$B8	23	21
\$31-\$38	7	7	\$B9-\$C0	24	21
\$39-\$40	8	8	\$C1-\$C8	25	22
\$41-\$48	9	8	\$C9-\$D0	26	23
\$49-\$50	10	9	\$D1-\$D8	27	24
\$51-\$58	11	10	\$D9-\$E0	28	25
\$59-\$60	12	11	\$E1-\$E8	29	26
\$61-\$68	13	12	\$E9-\$F0	30	27
\$69-\$70	14	14	\$F1-\$F8	31	28
\$71-\$78	15	14	\$F9-\$FF	32	28
\$79-\$80	16	15			

Bus inactivity limit: The bus inactivity limit field, bytes 4 and 5 of the disconnect/reconnect control parameters page, indicates the maximum time in 100-microsecond increments that the AppleCD SC will hold the bus busy until it disconnects. The AppleCD SC may round to its nearest capable value. A setting value of zero indicates that the AppleCD SC is allowed to maintain the bus busy indefinitely without handshakes until it determines to disconnect. The default value for the bus inactivity limit field is \$00.

Disconnect time limit: The disconnect time limit field, bytes 6 and 7 of the disconnect/reconnect control parameters page, indicates the minimum time in 100-microsecond increments that the AppleCD SC should remain disconnected until it attempts to reconnect. The AppleCD SC may round to its nearest capable value. A setting value of zero indicates that the AppleCD SC is allowed to reconnect immediately. The default value for disconnect time limit field is \$00.

Connect time limit: The connect time limit field, bytes 8 and 9 of the disconnect/reconnect control parameters page, indicates the maximum time in 100-microsecond increments that the AppleCD SC should remain connected until it attempts to disconnect. The AppleCD SC may round to its nearest capable value. A setting value of zero indicates that the AppleCD SC is allowed to remain connected indefinitely until it determines to attempt disconnection. The default value for connect time limit field is \$00.

Drive parameters page: The drive parameters page is used to pass parameters to the AppleCD SC drive controller. The format for the drive parameters page is shown in Figure 7-10. The fields in the drive parameters page are defined below.

Figure 7-10 Mode Select drive parameters page format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Rsvd	Rsvd	Page code =\$20					
	1	Page length in bytes = \$02							
	2	Reserved							
	3	Reserved				Inactivity time-out			

Page code: The page code field, bits 0–5 of byte 0, is set to \$20 for the drive parameters page.

Page length: The page length field, byte 1, is set to \$02, the length of the drive parameters page in bytes.

Inactivity timeout: The inactivity timeout field, bits 0–3 of byte 3, is used by the initiator to set the timeout value to spin-down the medium and turn off the optical pickup when the drive is inactive (for both data and audio operation). The drive will remain in the hold track state after completion of a seek or read operation during the inactivity timeout period. The timeout value is the value of this 4-bit field times 2 minutes. A \$00 in this field will turn on the drive infinitely. The default value of the inactivity timeout byte is \$05, which means the drive will turn off in approximately 10 minutes from the time when the drive enters a hold track state following a seek or read operation.

ModeSense (\$09)

Description: The ModeSense SmartPort control code executes the AppleCD SC SCSI Mode Sense command. The Mode Sense command provides a means for the AppleCD SC to report its medium, unit number, or peripheral device parameters to the initiator. It is a complementary command to the Mode Select command. The format for the Mode Sense command is shown in Figure 7-11. The SmartPort parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$04)
1	Unit number of the target AppleCD SC
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory location in which you provide the Mode Sense parameters to send to the AppleCD SC, and where you want the Mode Sense data returned.
6	Control code (\$09)
7	Transfer byte count

■ **Figure 7-11** Mode Sense command format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Sense data length							
	1	Medium type = \$00							
	2	Reserved							
	3	Block descriptor length = \$08							

ModeSense parameters: The parameters you need to specify in the `Mode Sense` request are the PCF (page control field), page code, and allocation length parameters described below.

PCF: The PCF field, (page control field) bits 7 and 6 of byte 2 in the AppleCD SC `Mode Sense` command block, defines the type of page parameter values to be returned. The possible bit values are shown in Table 7-7 and described below.

■ **Table 7-7** PCF bit values and definitions

Bit 7 & 6		Type of parameter values
0	0	Current values
0	1	Changeable values
1	0	Default values
1	1	Default values

- ◆ *Note:* Because the AppleCD SC cannot be written to, parameter values cannot be saved. A value of 1 in bit 7 and bit 6 of byte 2 will return default values.

Current values: The current values returned are either the parameters set in the last successful `Mode Select` command or the default values if a `Mode Select` command has not been executed.

Changeable values: The pages requested in bits 5–0 of byte 2 in the `Mode Sense` command block will return with a value set to 1 for the parameter bits that are allowed to be changed. Parameters that are not changeable will be set to 0. If any part of a field is changeable, all bits in that field are set to 1.

- ◆ *Note:* The page descriptor, as defined in this document, will always be returned even if none of the parameters is changeable within the page.

Default values: Pages requested will be returned to the initiator with the fields or bits set to the default values. Fields and bits not supported by the AppleCD SC shall be set to 0.

Page code: The page code, bits 0–5 of byte 2 in the Mode Sense command block, indicates which page(s) to return. The page codes are identical to the page codes defined earlier for the Mode Select command. The page codes are shown in Table 7-3.

Allocation length: The allocation length, byte 4 of the Mode Sense command, specifies the number of bytes that the initiator has allocated for returned Mode Sense data. An allocation length of zero indicates that no Mode Sense data shall be transferred. This condition shall not be considered as an error. Any other value indicates the maximum number of bytes that shall be transferred. The AppleCD SC shall terminate the data in phase when allocation length bytes have been transferred or when all available Mode Sense data has been transferred to the initiator, whichever is less.

Mode Sense data: The Mode Sense data contains a 4-byte header, followed by one 8-byte block descriptor, followed by zero or more pages of variable length. The Mode Sense header data format is shown in Figure 7-12. The block descriptor and page descriptor information is described below.

■ **Figure 7-12** Mode Sense header data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Sense data length							
	1	Medium type = \$00							
	2	Reserved							
	3	Block descriptor length = \$08							

Mode Sense header

Sense data length: The sense data length, byte 0, specifies the length in bytes of the subsequent `Mode Sense` data that is available to be transferred during the data in phase. The sense data length byte shall not include itself. If the allocation length specified in the command block is too small to transfer all the sense data, the sense data length shall not be adjusted to reflect the truncation.

Medium type: The medium type, byte 1, is \$00 for the AppleCD SC.

WP: The WP field (write-protected), bit 7 of byte 2, is reserved and set to \$00 for the AppleCD SC.

Block descriptor length: The block descriptor length specifies the length in bytes of one block descriptor. Because block descriptors are 8 bytes, this field should be set to \$08. The format for the block descriptor is shown in Figure 7-13. The block descriptor fields are described below.

■ **Figure 7-13** Mode Sense block descriptor data format

	Bit number							
	7	6	5	4	3	2	1	0
Byte number	0	Reserved						
	1	Number of blocks (MSB)						
	2	Number of blocks						
	3	Number of blocks (LSB)						
	4	Reserved						
	5	Block length (MSB)						
	6	Block length						
	7	Block length (LSB)						

Block descriptor

Number of blocks: The number of blocks fields, bytes 1 (MSB)–3(LSB), specifies the number of logical blocks on the medium that meet the block length in the block descriptor. A \$00 in all the number of blocks fields specifies all the logical blocks of the logical unit.

Block length: The block length fields, bytes 5 (MSB)–7(LSB), specify the length in bytes of the logical block size on the AppleCD SC. Block lengths of 256 (\$100), 512 (\$200), 1024 (\$400), 2048 (\$800), 2052 (\$804, optional), 2336 (\$920), 2340 (\$924), or 2352 (\$930, optional) bytes are supported by the AppleCD SC. The default block length is 2048 bytes.

Page descriptor

Pages: Pages are returned following zero or one block descriptor. Each page has a header that defines the page code and the page length. Following the header are page parameters. The page length value is the number of bytes that follow the page length byte, and does not include the header. The pages are defined under the `Mode Select` command. The page descriptor format is shown in Figure 7-14.

■ **Figure 7-14** Mode Sense page descriptor data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Rsvd	Rsvd	Page code					
	1	Page length in bytes							
	2 to n	Refer to the definition in ModeSelect							

You may request a particular page to be returned by the AppleCD SC by providing its code in byte 2 of the `Mode Sense` command block, described earlier.

ReadCapacity (\$0D)

Description: The ReadCapacity (\$0D) SmartPort control code executes the AppleCD SC SCSI Read Capacity command. The Read Capacity command requests that the AppleCD SC return the physical size of the currently mounted CD-ROM disc. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number of the target AppleCD SC
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address where you want the read capacity data returned.
6	Control code (\$0D)

The logical block address bytes, PMI (partial medium indicator) bit 0 byte 1, and RelAdr (relative address) bit 0 byte 8, are set to \$00 in this command.

The AppleCD SC returns eight bytes to the initiator during the data transfer phase. The first four bytes are the logical block address of the last valid logical block on the CD-ROM. The second four bytes are the block length in bytes of the logical blocks on this disc. The logical block address and the block length are based on the value of the block length set in the Mode Select command.

The default size of the block length is 2048 bytes. (Refer to the ANSI X3T9.2/82-2 Rev.17B specification for additional details.) The Read Capacity data format is shown in Figure 7-15.

■ **Figure 7-15** Read Capacity return data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Logical block address (MSB)							
	1	Logical block address							
	2	Logical block address							
	3	Logical block address (LSB)							
	4	Block length (MSB)							
	5	Block length							
	6	Block length							
	7	Block length (LSB)							

Inquiry (\$0F)

Description: The Inquiry (\$0F) SmartPort control code executes the AppleCD SC SCSI Inquiry command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address where you want the Inquiry data returned.
6	Control code (\$0F)

The Inquiry command returns information on the device type, firmware revision, conformance to various industry standards, and SCSI Command Set supported on the AppleCD SC. The requested information is sent to the host in the Inquiry return block shown in Figure 7-16.

■ **Figure 7-16** Inquiry command return data format

		Bit Number							
		7	6	5	4	3	2	1	0
Byte Number	0	device type (\$05)							
	1	rmb=1	device type qualifier (\$00)						
	2	ISO version		ECMA version			ANSI version		
	3	reserved				return data format = \$01			
	4	additional length (n)							
	5	vendor specific							
	6	reserved							
	7	reserved							
	8 15	vendor id (ASCII)							
	16 31	product id (ASCII)							
32 35	revision level (ASCII)								
36 55	vendor unique (command bit map)								

Peripheral device type: This field contains the SCSI standard device code of the AppleCD SC. It must be set to \$00 (direct access device).

RMB: This flag indicates whether or not the data recording medium of the target device is removable (1) or nonremovable (0).

Device type qualifier: This field contains a user-defined code that further specifies the target's device type. It must be set to \$00.

ISO version: This field contains the version-number code for the International Standards Organization. A nonzero value in this field indicates compliance with the ISO standard; the actual value loaded into this field must conform to the ISO definitions.

ECMA version: This field contains the version-number code for the European Computer Manufacturers Association. A nonzero value in this field indicates compliance with the ECMA standard; the actual value loaded into this field must conform to the ECMA definitions.

ANSI version: This field contains the version-number code for the ANSI SCSI standard. It must be set to \$01.

Reserved: All reserved fields must be set to \$00.

RDF: This field contains the response data format code for the information contained in the data block following the header, as shown in Table 7-8. The value for the RDF field is always \$01 for the AppleCD SC.

■ **Table 7-8** Response data format codes

Code	Response data format
\$00	Vendor specific
\$01	Common Command Set
\$02-\$0F	Reserved

Additional length: This field contains the length of the data block portion of the Inquiry command return block.

Vendor specific: This field contains vendor-specific information.

Vendor ID: This field contains an ASCII string identifying the target's vendor.

Product ID: This field contains an ASCII string identifying the target's product name and a model number.

Revision level: This field contains an ASCII string identifying the target's firmware revision level.

Command bitmap: This field contains the number of extensions supported by the `Reserve` command and a bitmap of the command set supported by the target.

The command bitmap is organized by SCSI command group codes. After each group code byte, the commands belonging to that group that are supported by the target are listed in a bitmap. The command list is four bytes long. Commands are mapped into it according to their command number, from lowest to highest, by either setting a bit (1) or clearing a bit (0). The lowest numbered command within a given group corresponds to bit 0 of that group's first list byte, while the highest-numbered command corresponds to bit 7 of that group's fourth list byte.

SetBlockSize (\$13)

Description: The `SetBlockSize ($13)` SmartPort control code sets the value for the number of bytes per block on the AppleCD SC. Legal block sizes in bytes are 512, 256, 1024, 2048, 2336, and 2340 for the AppleCD SC drive. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Block size (in bytes) (MSB-LSB)
6	Control code (\$13)

SetTimeout (\$14)

Description: The `SetTimeout ($14)` SmartPort control code sets the amount of time the initiator will wait for a response from the target before terminating the command (timeout). The timeout constant ranges from \$00-\$FF, and is loaded into the control call parameter list in byte 2. For this command, byte 3 of the parameter list must be set to 0. The default timeout constant is \$0C, for a timeout duration of approximately 10 seconds. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2	Timeout
3-5	Reserved (\$00)
6	Control code (\$14)

Seek (\$16)

Description: The `Seek ($16)` SmartPort control code executes the AppleCD SC SCSI `Seek Extended` command. After the command is completed, the optical pickup is in a hold track state for the duration of the inactivity timeout. Refer to the ANSI X3T9.2/82.2 Rev. 17B specification for additional details. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address that contains the logical address on the AppleCD SC to which you want to seek.
6	Control code (\$16)

StartUnit/StopUnit (\$18/\$19)

Description: These codes execute the AppleCD SC SCSI Start/StopUnit command. Each must be used separately; StartUnit (\$18) only starts the unit, while StopUnit (\$19) only stops it.

On StartUnit the drive will spin up, and set the laser beam, focus servo, and tracking servo into the On state. On StopUnit the drive will spin down, and the laser beam, focus servo, and tracking servo will be set into the Off state. The parameter list for these control codes is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Reserved (\$00)
6	Control code (\$18) = Start, (\$19) = Stop

PreventRemoval/AllowRemoval (\$1A/\$1B)

Description: These codes execute the AppleCD SC SCSI Prevent/Allow Removal command. The commands must be used separately. The parameter list for these control codes is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Reserved (\$00)
6	Control code (\$1A) = prevent, (\$1B) = allow

The `Prevent/Allow Removal` command requests that the target enable/disable the removal of the CD-ROM caddy in the AppleCD SC drive. Sending a `Prevent` call will inhibit the removal of the caddy with the `Eject` command or with the drive eject button. (The emergency pin hole eject mechanism is not overridden.) If the initiator issues an `Eject` command after a `Prevent` call, the `Eject` command will have Check Condition status with the sense key of illegal request and the additional sense code of \$80 (vendor unique) to prevent media removal. Prevention of the caddy removal condition will be terminated upon receipt of a Bus Device Reset message or a hard reset. Refer to the ANSI X3T9.2/82-2 Rev. 17B specification for details.

Verify (\$1C)

Description: The `Verify ($1C)` SmartPort control code executes the AppleCD SC SCSI `Verify` command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address that contains the address of the block on the AppleCD SC you want to verify. The block address on the CD-ROM is also a 4-byte field (MSB first).
6	Control code (\$1C)

RezeroUnit (\$1D)

Description: The `RezeroUnit ($1D)` SmartPort control code executes the AppleCD SC SCSI `RezeroUnit` command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Reserved (\$00)
6	Control code (\$1D)

The `RezeroUnit` command repositions the optical pickup at the starting address of the disc and enters the hold track state for the duration of the inactivity timeout. The drive will spin up if it is not currently spinning.

If you request `Disconnect`, disconnect will be performed before the seek operation and `reselect` will take place following the seek operation.

This command should not change the `Mode Select` parameters.

Patch1Call (\$1E)

Description: This code is used to patch one unsupported command to the command interpreter. It executes program code loaded into RAM bank 2 at \$C803. To use this call, you must write the program code, exactly as you want it to execute, and load it into bank 2 RAM starting at \$C803. Once you have loaded the program code, executing `Patch1Call` will automatically execute your command.

The parameter list for the `Patch1Call` control code is as follows:

Byte	Definition
0	Parameter list length (\$03)
1	Reserved (\$00)
2-5	Reserved (\$00)
6	Control code (\$1E)

- ◆ *Note:* Remember that your command structure is loaded in RAM; any reinitialization or loss of power will erase it.

SetNewSDAT (\$1F)

Description: This code forces the SCSI card to reinitialize the SDAT and DIBTAB device tables for all devices resident on the SCSI bus. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Reserved
6	Control code (\$1F)

AppleCD SC SmartPort audio control calls

The following SmartPort calls control the CD-Audio features of the AppleCD SC drive. The audio calls described below are not required for implementing computer processor sound on CD-ROM, but are required for accessing audio data on discs created with both CD-Audio tracks and stereo CDs (Philips/Sony *Red Book* standard).

AudioSearch (\$20)

Description: The AudioSearch (\$20) SmartPort control code executes the AppleCD SC SCSI AudioTrackSearch command. The parameter list for this control code is as follows

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer: This pointer points to the memory address that contains information described below.
6	Control code (\$20)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows.

Byte	Definition
0	Play flag (0=hold track after search, 1=play after search)
1	Play mode (\$00-\$FF)
2-5	Search address (LSB-MSB)
6	Type of search address (\$00-\$02)

The AudioSearch command provides a means for positioning the optical pickup at an address specified by the search address parameter. This command returns the status byte when the requested search address is found. If the search address is not found, or if the drive is not ready, or if the search address is not within an audio track, a Check Condition status will be returned.

A Play bit of 0 (set in the play flag parameter byte 0) indicates the drive will enter the hold track state (pause) when the search address is found. The music will then be played back when an `AudioPlay` command is issued. A play bit of 1 indicates that the drive will output the audio channels, specified in the play mode parameter, when the audio address is found.

The drive can accept and perform another command that will or will not terminate the current `AudioPlay` operation, as described in the `AudioPause` command description.

The Play mode parameter (byte 01, bits 3 to 0) specifies the play mode in binary notation as follows.

Play mode parameter	Meaning
0000	Play with muting on.
0001	Play right channel only through right channel.
0010	Play left channel through right channel only.
0011	Play left and right channels through right channel only.
0100	Play right channel through left channel only.
0101	Play right channel through left and right channel.
0110	Play right channel through left channel, left channel through right channel.
0111	Play right channel through left channel, left and right channels through right channel.
1000	Play left channel through left channel only.
1001	Play left channel through left channel, right channel through right channel, stereo.
1010	Play left channel through left and right channel.
1011	Play left channel through left channel, left and right channels through right channel.
1100	Play left and right channels through left channel.
1101	Play left and right channels through left channel, right channel through right channel.

**Play mode
parameter**

Meaning

1110	Play left and right channels through left channel, left channel through right channel.
1111	Play left and right channels through left channel, left and right channels through right channel.

The mnemonic for understanding the play mode bits is this: The right two bits (bits 1 and 0) stand for the speaker right channel output from left and right channels input; the left two bits (bits 3 and 2) stand for the speaker left channel output from left and right channels input.

The type parameter specifies the format of the search address used in the command. The values and definitions for address types that can be specified with the type parameter are shown in Table 7-9.

■ **Table 7-9** Values for search address type parameter

Type field	Description
\$00	Specifies search address in logical block address format.
\$01	Specifies search address in absolute minutes-seconds-blocks format.
\$02	Specifies search address in track number (TNO) format.
\$03	Reserved.

The `AudioSearch` type \$00 search address format, shown in Figure 7-17, specifies the logical block address at which the `AudioPlay` or `AudioPause` operation will begin.

- **Figure 7-17** AudioSearch address in type \$00 logical block format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	2	Logical block address (MSB)							
	3	Logical block address							
	4	Logical block address							
	5	Logical block address (LSB)							

The AudioSearch type \$01 search address parameter specifies the relative running time from the beginning of the disc in absolute minutes-seconds-frames. Byte 3, minutes, has a range of 0 to 99 in BCD. Byte 4, seconds, ranges from 0 to 59 in BCD. Byte 5, frames, has a range of 0 to 74 in BCD. The type \$01 format is shown in Figure 7-18.G177

- **Figure 7-18** AudioSearch address in type \$01 absolute minutes-seconds-frames format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	2	Reserved							
	3	CD-absolute address (minutes)							
	4	CD-absolute address (seconds)							
	5	CD-absolute address (frames)							

The `AudioSearch` type \$02 format specifies the track number (in BCD notation) at which the `AudioPlay` or `AudioPause` operation will begin. This track number has a range of 1 to 99 in BCD. The type \$02 format is shown in Figure 7-19.

■ **Figure 7-19** `AudioSearch` address in type \$02 track number format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	2	Reserved							
	3	Reserved							
	4	Reserved							
	5	Track number address							

AudioPlay (\$21)

Description: The `AudioPlay` (\$21) SmartPort control code executes the AppleCD SC SCSI `AudioPlay` command. The parameter list for this control code is as follows

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB)
6	Control code (\$21)

The buffer contains the control parameters required by the `Audio Play` SCSI command for proper execution, as follows.

Byte	Definition
0	Stop address flag (0=stop address in bytes 2-5; 1=play address)
1	Play mode (\$00-\$FF)
2-5	Playback address (LSB-MSB)
6	Type (\$00-\$02)

The `AudioPlay` command requests that the target position the optical pickup at the playback address specified in bytes 2 through 5 of the control parameter, then play audio through the audio channels specified in byte 1, the play mode parameter. This command returns the status byte when the playback address is found. If the playback address is not found, or if the drive is not ready, or if the address is not within an audio track, a Check Condition status will be returned.

The `AudioPlay` command is also used to release a pause (hold track) state after the execution of an `AudioSearch` command or a pause state after the execution of an `AudioPause` command. The `AudioPlay` command can also be used to set a particular completion address or play mode.

A stop address flag bit of 0 indicates that the Playback address is the starting address for the `AudioPlay` operation. In this case, the stop address should be set by a prior `AudioStop` command. A stop address flag bit of 1 indicates that the playback address is the stopping address of the `AudioPlay` operation. In this case, the starting address is set by a prior `AudioSearch` command. In both cases, if the stop address is larger than the end of the current audio track, the drive should play up to the stop address or until it encounters a data track. The stop address default is the last audio track after the drive is switched on or the media changed. If the stop address is smaller than the starting address, the drive will return a Check Condition with an illegal request sense key and should not play any audio. The minimum difference between the audio start address and the audio stop address is 25 blocks.

The AppleCD SC can accept and perform another command that will or will not terminate the current `AudioPlay` operation, as described under the `AudioPause` command description.

The play mode parameter (byte 01, bits 3 to 0) specifies the play mode in binary notation. (See the list of values provided for play mode in the `AudioSearch` address command.)

The type parameter specifies the format of the playback address used in the command. The type parameter is defined in Table 7-10.

■ **Table 7-10** Values for playback address type parameter

Type value	Description
\$00	Specifies playback address in logical block address format.
\$01	Specifies playback address in amin, asec, and aframe format.
\$02	Specifies playback address in track number (TNO) format.
\$03	Reserved.

The type \$00 parameter specifies the logical block address at which the `AudioPlay` operation will begin or end, depending on the setting of the stop address flag. *The logical block address format is identical to the logical block address format for the `AudioSearch` command.*

The type \$01 minutes, seconds, and frames fields specify the relative running time from the beginning of the disc. The amin field has a range of 0 to 99 in BCD. The asec field ranges from 0 to 59 in BCD. The aframe field has a range of 0 to 74 in BCD. *The format for this address type is identical to the minutes-seconds-blocks format for the `AudioSearch` command.*

The type \$02 track number field specifies the track in BCD notation at which the play operation will begin at the beginning of the specified track or end at the end of the specified track. This field has a range of 1 to 99 in BCD. *The format for the track number address type is identical to the track number format for the `AudioSearch` command.*

AudioPause (\$22)

Description: The `AudioPause` (\$22) SmartPort control code executes the AppleCD SC SCSI `AudioPause` command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB)
6	Control code (\$22)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows.

Byte	Definition
0	Pause flag (0)=pause, (1)=resume)

The `AudioPause` command temporarily stops the `AudioPlay` operation and places the optical head in the hold track state (keeps the disc at the same Q Subcode address).

A pause flag of 1 has the effect of temporarily stopping the `AudioPlay` operation with muting on and continuing a search of the same Q Subcode address. `AudioPause` is valid for use only when the `AudioPlay` is in progress after the `AudioSearch` or `AudioPlay` commands. A pause flag of 0 releases the `AudioPause` operation and resumes the `AudioPlay` operation at the same Q Subcode address just before the start of the `AudioPause` operation.

The `AudioPause` (pause flag = 1), `AudioPlay`, or `AudioScan` operation can be terminated with a `RezeroUnit`, `Read`, `Seek`, `Start/Stop Unit`, `Prevent/Allow Removal`, `Read Extended`, `ExtendedSeek`, `Verify`, `Eject`, `ReadHeader`, `AudioPlay` (stop address=0), `AudioPause`, `AudioScan`, `AudioSearch` command. The drive will terminate the current `AudioPause`, `AudioPlay`, or `AudioScan` operation and then perform the requested command.

The drive should accept and perform the `TestUnitReady`, `RequestSenseInquiry`, `Reserve`, `Release`, `ModeSelect`, `ModeSense`, `ReadCapacity`, `ReadTOC`, `ReadQSubcode`, `AudioStatus`, `AudioPlay` (stop address=1), or `AudioStop` command without terminating the current `AudioPause`, `AudioPlay`, or `AudioScan` operation in progress.

AudioStop (\$23)

Description: The `AudioStop` (\$23) SmartPort control code executes the AppleCD SC SCSI `AudioStop` command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB)
6	Control code (\$23)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows.

Byte	Definition
0-3	Stop address (MSB-LSB)
4	Type (\$00-\$02)

The `AudioStop` command specifies the stop address at which the `AudioPlay` operation will stop. The drive will spin down and the optical pickup is held around the area of the stop address.

If the stop address is 0 and the type parameter is \$00B, the `AudioStop` command will perform the same function as the `RezeroUnit` command.

The type parameter specifies the format of the stop address used in the command. The type parameter is defined in Table 7-11.

■ **Table 7-11** Values for stop address type parameter

Type value	Description
\$00	Specifies the stop address in logical block address format.
\$01	Specifies the stop address in absolute minutes-seconds-blocks format.
\$02	Specifies the stop address in track number (TNO) format.
\$03	Reserved.

The type \$00 logical block address fields specify the logical block address at which the play operation will stop. The format for this address type is identical to the logical block address format for the `AudioSearch` command.

The type \$01 minutes, seconds, and frames fields specify the relative running time from the beginning of the disc. The minutes field has a range of 0 to 99 in BCD. The seconds field ranges from 0 to 59 in BCD. The frames field has a range of 0 to 74 in BCD. The format for this address type is identical to the minutes, seconds, and frames address format for the `AudioSearch` command.

The type \$02 track number field specifies the track in BCD notation at which the play operation will stop. This field has a range of 1 to 99 in BCD. The format for this address type is identical to the track number address format for the `AudioSearch` command.

AudioStatus (\$24)

Description: The AudioStatus (\$24) SmartPort control code executes the AppleCD SC SCSI AudioStatus command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address to which you want the audio status information returned.
6	Control code (\$24)

This AudioStatus command transfers the current AudioPlay status and the starting Q Subcode address of next track.

The data returned by the AudioStatus command is returned in 6-byte format, shown in Figure 7-20. The status field, byte 0, contains one of the values defined in Table 7-12.

When the AppleCD SC is not in the Ready state, a Check Condition is returned.

■ **Figure 7-20** AudioStatus data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Status							
	1	Reserved				Play mode			
	2	Reserved				Control			
	3	Next track starting address (minutes)							
	4	Next track starting address (seconds)							
	5	Next track starting address (frames)							

■ **Table 7-12** AudioStatus data field byte values and definitions

Byte value	Description
\$00	AudioPlay operation in progress following execution of
	AudioSearch command (play bit = 1) or AudioPlay command.
\$01	Pause (hold track state) operation in progress.
\$02	Muting On operation in progress after execution of
	AudioSearch command or AudioPlay command.
\$03	AudioPlay completion status.
\$04	Error occurred during AudioPlay operation.
\$05	AudioPlay operation not requested.

Play mode (Byte 01, bits 3 to 0) specifies the play mode in BCD. (See the list of values provided for the play mode parameter in the AudioSearch command.)

Bits 0–3 of byte 2 contain the control field of the next track. The bit values are defined in Table 7-13. (The X indicates that any value may be set for that bit.)

■ **Table 7-13** Control field bit values and definitions

3	Bits			Definition
	2	1	0	
0	0	X	0	Two audio channels without preemphasis
0	0	X	1	Two audio channels with preemphasis
1	0	X	0	Four audio channels without preemphasis
1	0	X	1	Four audio channels with preemphasis
0	1	X	0	Data track
0	1	X	1	Reserved
1	1	X	X	Reserved
X	X	0	X	Digital copy protected
X	X	1	X	Digital copy permitted

AudioScan (\$25)

Description: The AudioScan (\$25) SmartPort control code executes the AppleCD SC SCSI AudioScan command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2–5	Buffer pointer (LSB–MSB)
6	Control code (\$25)

The buffer pointer, bytes 2–5 in the parameter list, points to the control parameters required by the AudioScan command for proper execution. The parameters are as follows.

Byte	Definition
0	Direction flag (0=forward, 1=reverse)
1	Reserved (\$00)
2-5	Scan start address (LSB-MSB)
6	Type (\$00-\$02)

The `AudioScan` command requests a fast-forward or fast-reverse scan operation starting from the scan start address.

The drive can accept and perform another command which will or will not terminate the current `AudioScan` operation, as described in the `AudioPause` command description.

A direction bit of 0 indicates a fast-forward operation. A direction bit of 1 indicates a fast-reversed operation.

The scan start address specifies the address at which to begin the fast scan. The address definition varies with the type parameter.

The type parameter specifies the format of the scan start address used in the command. The type parameter is defined in Table 7-14.

■ **Table 7-14** Values for scan start address type parameter

Type value	Description
\$00	Specifies that scan start address is in logical block address format.
\$01	Specifies that scan start address is in absolute minutes-seconds-frames format.
\$02	Specifies that scan start address is in track number format.
\$03	Reserved.

Scan start address type \$00 specifies the logical block start address at which the `AudioScan` operation will begin. The format for specifying the logical block address is shown in Figure 7-21.

- **Figure 7-21** Scan start address type \$00 logical block format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	2	Logical block address (MSB)							
	3	Logical block address							
	4	Logical block address							
	5	Logical block address (LSB)							

Scan start address type \$01 specifies the relative running time from the beginning of the disc at which the `AudioScan` operation will begin. The absolute minutes field has a range of 0 to 99 in BCD. The absolute seconds field ranges from 0 to 59 in BCD. The absolute frames field has a range of 0 to 74 in BCD. The format for specifying the absolute minutes, seconds, and frames address is shown in Figure 7-22.

- **Figure 7-22** Scan start address type \$01 in absolute minutes-seconds-frames format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	2	Reserved \$00							
	3	CD-absolute address (minutes)							
	4	CD-absolute address (seconds)							
	5	CD-absolute address (frames)							

The scan start address Type \$02 specifies the track number in BCD notation at which the `AudioScan` operation will begin. This field has a range of 1 to 99 in BCD. The format is shown in Figure 7-23.

■ **Figure 7-23** Scan start address type \$02 track number format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	2	Reserved							
	3	Reserved							
	4	Reserved							
	5	Track number (in binary coded decimal)							

ReadTOC (\$27)

Description: The `ReadTOC ($27)` SmartPort control code executes the AppleCD SC SCSI `ReadTOC` command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB)
6	Control code (\$27)

The buffer contains the control parameters required by the SCSI command for proper execution, as follows

Byte	Definition
0	Track number
1-2	Allocation length (LSB-MSB)
3	Type (\$00-\$02)

This command transfers TOC (table of contents) data from the CD audio disc to the initiator.

The track number parameter specifies the starting track number in BCD for the cases when the type has the value of \$02. The starting track ranges from 1 to 99 in BCD. The track number byte is reserved for type values of \$00, \$01, or \$03.

If you specify a track number larger than the last track number, or specify a track number smaller than the first track number, the Check Condition status is returned.

The allocation length parameter specifies the number of bytes that the initiator has allocated for TOC returned data. An allocation length of 0 indicates that no data will be transferred. Any other value indicates the maximum number of bytes that will be transferred. The AppleCD SC terminates the data-in phase when allocation length bytes have been transferred or when all available data has been transferred to the initiator, whichever is less.

The type parameter for the `ReadTOC` command specifies the format in which the table-of-contents information will be returned. The type parameter values are defined in Table 7-15.

■ **Table 7-15** Values for ReadTOC type parameter

Type value	Description
\$00	Transfers the first and last user track number in BCD.
\$01	Transfers the starting address of the lead out area in BCD minutes-seconds-frames format.
\$02	Transfers the starting address of each track for a range of tracks, starting from the track specified in the track number byte, in ascending sequential order until the number of bytes specified in the allocation length have been transferred or until all available data has been transferred to the initiator, whichever is less.
\$03	Reserved.

The format for type \$00 TOC data is shown in Figure 7-24.

■ **Figure 7-24** ReadTOC type \$00 data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	First user track number (in binary coded decimal)							
	1	Last user track number (in binary coded decimal)							
	2	Reserved							
	3	Reserved							

The type \$01 format transfers the starting address of the lead-out area in BCD minutes-seconds-frames format, as shown in Figure 7-25.

■ **Figure 7-25** ReadTOC type \$01 data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Lead-out area starting address in BCD (minutes)							
	1	Lead-out area starting address in BCD (seconds)							
	2	Lead-out area starting address in BCD (frames)							
	3	Reserved							

The type \$02 format transfers the starting address information (consisting of control field, minutes, seconds, and frames) of each track for a range of tracks, starting from the track specified in the track number byte, in ascending sequential order until the number of bytes specified in the allocation length have been transferred or until all available data has been transferred to the initiator, whichever is less. The format for type \$00 TOC data is shown in Figure 7-26.

■ **Figure 7-26** ReadTOC type \$02 data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Reserved				Control field of specified track			
	1	Starting point of a specified track (minutes in BCD)							
	2	Starting point of a specified track (seconds in BCD)							
	3	Starting point of a specified track (frames in BCD)							

The above 4-byte starting address information is repeated for each track in a range of tracks.

The following is the `ReadTOC` strategy after a power up or disc change. In addition, the AppleCD SC does not read the TOC again after a hard reset from the SCSI bus.

1. The AppleCD SC drive returns the Check Condition status when any command is sent from the host during the TOC read.
2. The AppleCD SC drive returns a sense key Not Ready and an additional sense code of \$B7 (`ReadTOC` in progress) when host makes a `RequestSense` call after a Check Condition as stated in step 1 above.
3. If the drive is unable to read the TOC in the maximum interval allowed by the *Yellow Book* standard, the drive gives a Check Condition followed by a sense key of Not Ready and an additional sense code of \$B1 (unable to recover TOC).
4. The `ReadTOC` command will not force the drive to read the TOC unless there has been a disc change or a restart.

ReadQSubcode (\$28)

Description: The `ReadQSubcode` (\$28) SmartPort control code executes the AppleCD SC SCSI `ReadQSubcode` command. The parameter list for this control code is as follows.

Byte	Definition
0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB) points to the address in which you provide the allocation length when the command is sent, and the address to which you want the Q Subcode information returned
6	Control code (\$28)

The `ReadQSubcode` command transfers the current Q Subcode address loaded from either an audio or a data track to the initiator.

When the AppleCD SC is not in the ready state, a Check Condition is returned.

The allocation length parameter specifies the number of bytes that you want allocated for returned data. An allocation length of zero indicates that no data will be transferred. Any other value indicates the maximum number of bytes allocated for the data that will be transferred. The AppleCD SC terminates the data-in phase to the initiator when allocation length bytes have been transferred or when all available data has been transferred, whichever is less. The format of the Q Subcode data is shown in Figure 7-27.

■ **Figure 7-27** Q Subcode data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Reserved				Control field			
	1	CD Q subcode address data (track number)							
	2	CD Q subcode address data (Index)							
	3	CD Q subcode address data (minutes)							
	4	CD Q subcode address data (seconds)							
	5	CD Q subcode address data (frames)							
	6	CD Q subcode address data (Aminutes)							
	7	CD Q subcode address data (Aseconds)							
	8	CD Q subcode address data (Aframes)							

The track number specifies the current track number in BCD notation. The value of this field has a range of 01 to 99.

The control field is defined in Table 7-16. (The X in the bit fields indicates that any value may be set for that bit.)

The index number specifies the index number in the current track.

The minutes, seconds, and frames fields specify the relative running time from the beginning of the track.

The absolute minutes, seconds, and frames fields specify the relative running time from the beginning of the disc.

■ **Table 7-16** Control field bit values and definitions

Bits				Definition
3	2	1	0	
0	0	X	0	Two audio channels without preemphasis
0	0	X	1	Two audio channels with preemphasis
1	0	X	0	Four audio channels without preemphasis
1	0	X	1	Four audio channels with preemphasis
0	1	X	0	Data track
0	1	X	1	Reserved
1	1	X	X	Reserved
X	X	0	X	Digital copy protected
X	X	1	X	Digital copy permitted

ReadHeader (\$29)

Description: The ReadHeader (\$29) SmartPort control code executes the AppleCD SC SCSI ReadHeader command. The parameter list for this control code is as follows.

Byte	Definition
------	------------

0	Parameter list length (\$03)
1	SmartPort unit number
2-5	Buffer pointer (LSB-MSB): This pointer points to the memory address in which you provide the logical block address on the CD-ROM that you want header information from and to which you want the header information returned.
6	Control code (\$29)

The four-byte buffer contains the control parameters required by the SCSI command for proper execution, as follows.

Byte	Definition
------	------------

0-3	Logical block address (MSB-LSB)
-----	---------------------------------

The `ReadHeader` command returns 4 bytes of header information for the specified logical block address on the CD-ROM. If the logical block address is within an audio track, a Check Condition will be reported in the status phase. The data format of the 4-byte header is shown in Figure 7-28.

The first three bytes, bytes 0-2, of the returned header data contain the address of the specified logical block in absolute minutes-seconds-frames format and the last byte, byte 3, contains the mode of the block.

■ **Figure 7-28** Header data format

		Bit number							
		7	6	5	4	3	2	1	0
Byte number	0	Address in absolute minutes							
	1	Address in absolute seconds							
	2	Address in absolute frames							
	3	Mode of block							

Appendix A **Sense Code Definitions**

The following additional sense codes, which are specific to the AppleCD SC, are returned in byte 12 of the extended sense data when the `RequestSense` SCSI command is used. The `RequestSense` command can be called through the Macintosh AppleCD SC `GetSense` (99) call or the Apple II SmartPort `RequestSense` (\$06) call. ■

BLANK
(Blind Folio)

■ **Table A-1** Additional sense codes

Additional sense code	Definition	Related sense keys
\$00	No additional sense data	
\$01	Reserved	
\$02	Error occurred during Seek operation	\$3, \$4
\$03	Reserved	
\$04	Drive not ready (off-line)	\$2
\$05–\$07	Reserved	
\$08	Logical unit communication failure	\$4
\$09	Track servo error	\$4
\$0A–\$10	Reserved	
\$11	Layered error correction code (LECC); uncorrectable data error	\$3
\$12–\$16	Reserved	
\$17	Cross Interleaved Reed-Soloman Coding (CIRC) recovered data error (LECC off)	\$1
\$18	LECC recovered data error (LECC on)	\$1
\$19–\$1F	Reserved	
\$20	Invalid command code	\$5
\$21	Illegal logical block address	\$5
\$22	Illegal function for device type	\$5
\$23	Reserved	
\$24	Illegal field in command block (other than command number or logical block address)	
\$25	Invalid logical unit number	\$5
\$26	Invalid field in parameter list	\$5
\$27	Reserved	
\$28	Medium changed (detected by medium insertion)	\$6
\$29	Power-on, reset, or SCSI bus device reset occurred	\$6
\$2A	Mode Select parameters changed	\$6

■ **Table A-1** Additional sense codes (continued)

Additional sense code	Definition	Related sense keys
\$2B-\$3F	Reserved	
\$40	SCSI controller data buffer failure	\$4
\$41	Drive internal bus data path failure	\$4
\$42	Power-on diagnostic error	\$4
\$43	Message rejection error	\$4, \$B
\$44	Drive controller error	\$4
\$45	Reselect failed	\$4
\$46	Reserved	
\$47	SCSI parity error (\$4)	
\$48	Host-detected error occurred	\$4, \$B
\$49	Inappropriate/illegal message	\$4, \$B
\$4A-\$7F	Reserved	
\$80	Prevent Medium Removal is set	\$5
\$81	Logical unit is reserved	\$5
\$82	End-of-user-area encountered for this track (data/audio mode change only)	\$5
\$83	Overlapped commands attempted	\$5
\$84	Illegal mode for track (for example, audio command for data track)	\$5
\$85-\$AF	Vendor specific	
\$B0	No medium in drive	\$2
\$B1	Unable to recover TOC (table of contents)	\$2
\$B2	Medium load/eject failure	\$2
\$B3	CIRC unrecoverable data error (LECC off)	\$3
\$B4	Focus servo failed	\$4
\$B5	Spindle servo failed	\$4
\$B6	Medium load mechanism failed	\$4
\$B7-\$FF	Vendor specific	

The following definitions apply to errors reported by the AppleCD SC.

Error	Definition
CIRC recovered data error	This error is defined as a data block for which the mode 2 flag was set, but on a subsequent read retry operation it was not set. The number of the subsequent read operations is limited to the read retry count (byte 3 of the error recovery parameters page). LECC is not used in this case.
CIRC unrecovered data error	This error is defined as a data block for which the mode 2 flag was set on all read operations up to the read retry count. LECC is not used in this case.
LECC recovered data error	This error is defined as a data block for which mode 2 is asserted but the LECC was able to correct the error within the read retry count.
LECC uncorrectable data error	This error is defined as a data block that could not be corrected by the LECC within the read retry count.

BLANK

(Blind folio)

Glossary

access time: The time it takes to find, retrieve, and display a piece of recorded information. Access time is usually measured as the worst case: the longest time to get from one piece of information to another.

address: The position on a computer storage medium at which a given piece of information is recorded. The position is identified by a numeric code. It can be described generally (the relative address) or specifically (the absolute address).

algorithm: A plan of the precise sequence of steps needed to accomplish a task.

aliasing: The effect that produces sharply noticeable inaccuracies in the recreation of any sound, line, or surface, caused by too low sampling rate.

authoring: The preparation of a computer program with an authoring language or authoring system. For example, HyperCard provides authoring tools that allow people without formal computer programming experience to prepare applications for computer-based systems.

BCD: Acronym for *binary-coded decimal*—a base-10 number represented by the digits 0 and 1. For example, the numbers 0, 1, 2, 3, and 4 in base-10 notation are 0, 1, 01, 11, and 100 in binary notation.

Boolean operator: An operator such as AND, OR, NOT, EXCEPT, IF, or THEN that allows you to logically refine your search for information.

buffer: A block of memory used to store data temporarily.

CAD: Acronym for *computer-aided design*.

CD: Compact disc—a format that digitally encodes sound on a 12 cm laserdisc. The sound is decoded by optical laser, which provides exceptional high-quality audio playback.

CD-I: Acronym for *compact disc interactive*—a new CD technology that will be released in the future. It stores audio, still graphics, and motion video.

CD-IV: Acronym for *compact disc interactive video*—a proposed extension to CD-I with full-motion video capabilities.

delivery system: The combination of hardware and software required to deliver an interactive video presentation. It may consist of a computer, a monitor, and CD-ROM player, or be extended to include multiple monitors and various magnetic and optical peripherals.

disc: The name commonly associated with *reflective optical videodisc* and CD-ROM.

DV-I: Acronym for *digital video interactive*. A system under development that uses advanced compression techniques to allow interactive processing and storage of over one hour of full-motion, full-frame, low-resolution video.

index system: A list that provides the retrieval software with the locations of all the relevant data on the disc. The index can help to provide additional logical links to data on the disc. This list is created with inversion software and placed on the CD-ROM with the information to which it points.

interactive video: The combination of a computer program and video program running simultaneously under user control. Interactive video allows the user to affect the way the program evolves. In contrast, television programs or movies are linear video presentations.

inversion software: The software used to build an index system from all of the data that has been collected and organized for placement on the CD-ROM. Data may consist of graphics, text, HyperCard stacks, sound, animated segments, and any other form of information that can be stored on a computer.

Green Book: The Philips/Sony book of standards for CD-I technology.

mode 1: One of two encoding specifications for the 2352-byte CD-ROM block structure. Mode 1 is made up of 2048 bytes of user data with the remaining bytes comprising error detection and correction, block addresses, headers, and synchronization information. Mode 1 is the most common mode for CD-ROM block structure because it provides extensive error correction for user data. Compare **mode 2**.

mode 2: One of two encoding specifications for the 2352-byte CD-ROM block structure. Mode 2 is made up of 2336 bytes of user data with the remaining bytes comprising header and synchronization information. Mode 2 is useful for instances where data integrity is not critical. A possible use for mode 2 CD-ROM might include storage of graphics images.

optical disc: A recording medium for video and digital information. Optical disc technology allows large amounts of information to be stored on a single disc.

Red Book: The Philips/Sony book of standards for CD-Audio technology.

reflective optical videodisc: The format that uses lasers to record and play back a videodisc. This format is capable of playing audio and full-motion video.

ResEdit: A program supplied by Apple Computer, Inc. for the management and manipulation of resources. With this program, resources can be defined, edited, and moved between applications.

resource ID: A unique identification number within a type. Each resource in an application has associated with it a resource type and resource ID.

resources: A unit or collection of information used by an application program. Resources are generally stored in the resource fork of a file and loaded into memory when needed by the application. Most objects in the Macintosh environment are resources.

resource type: A four-character code defining the classification of a resource. Apple Computer, Inc. maintains a registry of these codes to ensure that no conflicts arise between applications.

SCSI device: A peripheral device, such as an AppleCD SC, that uses the *Small Computer System Interface (SCSI)*, which is an industry-standard interface providing high-speed access.

videodisc: A generic term used to describe the video storage medium on disc. Videodisc systems can be optical and capacitant. Optical systems use laser optics to read the disc; they are reflective or transmissive systems. Capacitance systems use a sensor or stylus to read grooved or grooveless discs.

WORM: Acronym for *Write Once Read Many*—a writable version of optical media. WORM discs are generally larger in diameter than CD-ROM discs and require WORM drives for recording and playback. An alternative to magnetic tape for laying out and delivering source for the CD-ROM premastering process.

Yellow Book: The Philips/Sony book of standards for CD-ROM technology.

BLANK

(Blind Folio)

Index

A

absolute minutes-seconds-frames
format 121
accessing audio data 121-135,
177-199
address for audio search 121-122,
130, 177-181
aliasing 85
AND operator 71
APause (105) call 132
APDA xvii, 26
APlay (104) call 131
AppleCD SC. *See* CD-ROM or
specific topic
AppleCD SC device driver 111-138
calls 115-138
controlling from application
111-112
AppleCD SC drive 6-7
AppleCD SC SCSI commands
115-199. *See also* SCSI
Apple Common Command Set
codes 148-149
AppleShare 105
AppleSignature field 42
Apple II AppleCD SC SmartPort
calls 139-200. *See also*
SmartPort audio control
calls; SmartPort control calls
Apple II family, developing for 26
Apple IIe, formats for 38
Apple IIGS
cross-development system 26
High Sierra files on 38
ISO 9660 files on 38-47

Apple IIGS extended format for
SmartPort calls 142
Apple IIGS Programmer's
Workshop (APW) 26
applications, multimedia 78
AScan (108) call 135
associated files 30, 47
AStatus (107) call 134
AStop (106) call 133
ATrkSearch (103) call 130
audience, defining 11
audio. *See* CD-Audio; sound
Audio CD Access file 31-33
audio control calls. *See* Macintosh
audio control calls;
SmartPort audio control
calls
AudioPause (\$22) call 184
Audio Pause command 184
AudioPlay (\$21) call
181-188
Audio Play command 181
AudioScan (\$25) call 189
Audio Scan command 189
AudioSearch address 180
AudioSearch (\$20) call
177
AudioStatus (\$24) call
187-189
Audio Status command
187
AudioStop (\$23) call 185
Audio Stop command 185
Audio Track Search
command 177
authoring CD-ROM 81

B

background, HyperCard 51-53
block descriptor
Mode Select command
146-147
Mode Sense command
166
block level CD-ROM disc 39
Boolean AND 71
build environment 81
button, HyperCard 52-53

C

calls. *See* Macintosh audio,
control, *and* CD-ROM
status calls; SmartPort
audio control calls;
SmartPort control calls; or
specific call
capturing processor sound 79
card, HyperCard 52
CD-Audio 76-78, 88-103. *See also*
Macintosh audio control
calls; SmartPort audio
control calls; sound
applications, use in 80-83
authoring 81
controlling AppleCD SC audio
88-103
data structure 121-122
master tape 81
PCM 501K tape 81
and processor sound compared
76-78
sample code 89-103

- simulation during testing 83
- test environment 83
- time gap 83, 121
- CD-ROM: *See also specific topic*
 - advantages of 2, 5
 - creating a disc 9-26
 - data preparation 16-18
 - definition 2
 - delivery systems 14
 - disc mastering 10, 20-22
 - features of 2, 6
 - HyperCard and 49-73
 - icon on desktop 34
 - logical formatting 19
 - mastering process 21-22
 - packaging 13, 24
 - processor sound for 76-79
 - product design 11-13
 - replication of disks 10, 23
 - software development 15-16, 24-26
 - standards for file systems 28-30
 - testing 23
 - uses of 3
- CD-ROM device driver. *See* AppleCD SC device driver
- CD-ROM discs. *See* discs
- character sets for filenames 30
- Check Condition status 150-154
- CIRC data recovery 157, 204
- coding data 17
- collecting data 17
- command byte in SmartPort call 140
- compact disc read-only memory. *See* CD-ROM or *specific topic*
- compressing data 17
- control calls. *See* Macintosh audio
 - control calls; Macintosh
 - control calls; Macintosh
 - status calls; SmartPort
 - audio control calls;
 - SmartPort control calls
- converting information from other media 12, 17-18

- C programming language 25-26, 50
- creating a CD-ROM 9-26
- csParam code 114

D

- data compression 17
- data organization 122
- data preparation 10, 16-18
 - coding 17
- data-retrieval engine, HyperCard as 54-73
- delivery system, defining 14
- design, product 11
- developer assistance xvi
- device driver 15. *See also* AppleCD SC device driver
- Device Manager routines 113
- device resources 15
- DIBTAB device table 176
- digitizers 86
- digitizing sound 79
- directory record 30, 42
- discs
 - identification of 31-33
 - mastering 10, 20-22
 - replication 10, 23
 - table of contents of 127, 193
 - testing 23
- disconnect/reconnect control page 158-161
- DiskStatus (8) call 136
- DriveIcon (22) call 117
- DriveInfo (23) call 118
- drive parameters page 160
- driver. *See* AppleCD SC device driver

E

- Eject (\$04) call 142
- Eject (7) call 116
- Eject command 142
- Eject/Initialize dialog box 33

- engine, retrieval 12, 15, 54-73
- equipment for product development 24-26
- error definitions 204
- error recovery page 150-157
 - default value 155
 - format of 151
 - page byte values 154-155
 - procedures 155-157
- exact match search 71
- extended format for SmartPort calls 142
- extended sense data format 143
- external commands, HyperCard 58, 62

F

- fields, HyperCard 53
- file format 19
 - choosing 19
 - comparison of 39
 - HFS 31-33, 39, 47
 - High Sierra 28, 33-39
 - ISO 9660 28-47
 - ProDOS 19, 40, 45-46
- filename transformation, ProDOS 45-47
- files
 - associated 30, 47
 - Audio CD Access 31, 32
 - hidden 30
 - High Sierra File Access 31, 32, 34
 - ISO 9660 File Access 31, 32, 34
 - naming conventions 30-31
 - Foreign File Access software 31-36
- format. *See* file format

G

- gap time, inter-track 83, 121
- GetSense (99) call 138, 201
- GetSize (98) call 137
- graphical interface 59-61

H

- handshaking method 113
- hardware
 - required for Apple II 26
 - required for Macintosh 24
 - sound-capturing 86
- hidden file 30
- hierarchical file system (HFS)
 - 31-33, 39, 47
- High Sierra File Access 31-33
- High Sierra file system 28, 33-39
 - Apple II computers and 38
 - compared with HFS 39
 - Macintosh desktop and 33-36
- HyperCard 49-73, 84-85
 - advantages of 50
 - background 51-53
 - button 52-53
 - card 52
 - existing CD-ROM database and 63-65
 - external command 58, 62
 - field 53
 - Find function 71
 - graphical interface 59-61
 - increasing performance of 71-72
 - interface considerations 59-70
 - international issues 70
 - inversion software 54-58
 - layout and design 69
 - limitations of 71-73
 - localization 70
 - locked stack 72
 - object 51-53
 - programming strategy 68
 - as retrieval engine 54-73
 - scripting language 51
 - search mechanism 71
 - 'snd ' resource type 85
 - sound and 84-87
 - stack 51-52
 - testing interface design 69
 - UserModify 72
 - XCMD 58, 62
- HyperTalk 51

I, J, K

- icon, CD-ROM 34
- indexing data 18
- Initialization failed dialog box 33
- Inquiry command 168
- Inquiry (\$0F) call 168-171
- interface 13, 15
 - HyperCard 59-70
 - non-Macintosh 62
 - SmartPort 139-142
- international issues
 - 28-29, 70
- inter-track gap time 83, 121
- inversion software 54-58
- I/O parameter block 113
- ISO 9660 file system 28-47
 - Apple extension for 40-47
 - Apple II computer and 38
 - associated files 30, 47
 - compared with High Sierra and HFS 39
 - design goals 29
 - directory record 30, 42-44
 - file naming conventions 30-31
 - HFS filename transformation 47
 - Macintosh and 31-38, 40-44, 47
 - ProDOS filename transformation 45-46
 - protocol identifier 41

L

- layout, HyperCard 69
- LECC recovered data 155-157, 204
- licensing information 12
- localization 70
- locked stacks 72
- logical block addressing format 121
- logical formatting 19

M

- MacApp 25
- Macintosh
 - Control routine 114
 - developing for 24-25
 - device driver calls 115-138
 - Foreign File Access 31-36
 - ISO 9660 and 31-38, 40-44, 47
- Macintosh audio control calls 121-135
 - APause (105) 132
 - APlay (104) 131
 - AScan (108) 135
 - AStatus (107) 134
 - AStop (106) 133
 - ATrkSearch (103) 130
 - ReadHead (102) 129
 - ReadQ (101) 128
 - ReadTOC (100) 127
- Macintosh CD-ROM status calls 136-138
 - DiskStatus (8) 136
 - GetSense (99) 138
 - GetSize (98) 137
 - WhoIsThere (97) 137
- Macintosh control calls 116-120
 - DriveInfo (23) 118
 - DriveIcon (22) 117
 - Eject (7) 116
 - MediaIcon (21) 117
 - RSwitch (78) 119
 - SCSI (77) 118
 - UserEject (80) 120
 - VarBlk (79) 119
- Macintosh Programmer's Workshop (MPW) 25
 - MPW C 25
 - MPW Pascal 25
- manuals, needed for development 25
- mastering the disc 10, 21-22
- MediaIcon (21) call 117

minutes-seconds-frames format 121
minute unit 121
ModeSelect (\$08) call 145-161
Mode Select command 145
ModeSense (\$09) call 162-166
Mode Sense command 162

N

naming server volumes 106
networks 105-107
Nyquist frequency 85

O

object, HyperCard 51-53
OS routines, Macintosh 112
organization of data on discs 122

P

packaging 13, 24
page data types 149
page descriptors 148, 166
 Mode Select command 148
 Mode Sense command 166
parameter block 115
parameter list, SmartPort call 140
Pascal 25
Patch1Call (\$1E) call 175
pause 188
PCM 501K format 81
Philips/Sony *Red Book* 121
Philips/Sony *Yellow Book* 29, 121
playing back sound 79
Play mode parameter 123, 178-179
pointer in SmartPort call 140
premastering tapes 20
preparing data 16-18
PreventRemoval/Allow
 Removal (\$1A/\$1B) calls 173

212 Index

Primary Volume Descriptor 41
processor sound 76-79
 compared with CD-Audio 76-77
 explanation of process 79
 using in applications 78
ProDOS filename transformation 45-46
ProDOS SmartPort environment 139
product design 11-13
product development 11-26
programming languages for development 25-26

Q

Q Subcode data format 128, 197

R

ReadCapacity (\$0D) call 167-168
Read Capacity command 167
ReadHead (102) call 129
ReadHeader (\$29) call 198-199
Read Header command 198
ReadQ (101) call 128
ReadQSubcode (\$28) call 196-198
Read Q Subcode command 196
ReadTOC (\$27) 192-195
ReadTOC (100) call 127
Read TOC command 192
ReadTOC type parameter 194-196
 Red Book 121
replication of discs 10, 23
RequestSense (\$06) call 143-144, 201
Request Sense command 143, 201
 return sense data format 144
resource fork 85
retrieval software 12, 15

 porting to Macintosh 62
 using HyperCard as 54-73
RezeroUnit (\$1D) call 175
Rezero Unit command 175
routines
 Macintosh OS 112
 Macintosh OS Device Manager 113
 SCSI 113
RSwitch (78) call 119
run-time retrieval software 58

S

sample program for audio 88-102
sampling rates 76, 85
sampling sound 79
scan start address 190-192
scripting language 51
SCSI
 AppleCD SC calls 113-114, 136
SDAT device table 176
search calls, audio
 ATrkSearch (103) 130
 AudioSearch (\$20) 177
search interface 59-60
search mechanism, HyperCard 71
Seek (\$16) call 172
Seek command 172
sense codes 143-144, 202-203
server volume, naming 106
SetBlockSize (\$13) call 171
SetNewSDAT (\$1F) call 176
SetTimeout (\$14) call 172
SmartPort audio control calls,
 AppleCD SC 177-201
 AudioPause (\$22) 184
 AudioPlay (\$21) 181-183
 AudioScan (\$25) 189-192
 AudioSearch (\$20) 177-180
 AudioStatus (\$24) 187-189

- AudioStop (\$23) 185-186
- ReadHeader (\$29) 198-199
- ReadQSubcode (\$28) 196-198
- ReadTOC (\$27) 192-195
- SmartPort command interpreter 139-140
- SmartPort control calls. *See also* SmartPort audio control calls
 - Eject (\$04) 142
 - Inquiry (\$0F) 168-171
 - ModeSelect (\$08) 145-161
 - ModeSense (\$09) 162-166
 - Patch1Call (\$1E) 175
 - PreventRemoval /AllowRemoval (\$1A/\$1B) 173
 - ReadCapacity (\$0D) 167-168
 - RequestSense (\$06) 143-144, 201
 - RezeroUnit (\$1D) 175
 - Seek (\$16) 172
 - SetBlockSize (\$13) 171
 - SetNewSDAT (\$1F) 176
 - SetTimeout (\$14) 172
 - StartUnit/StopUnit (\$18/\$19) 173
 - TestUnitReady (\$05) 143
 - Verify (\$1C) 174
- SmartPort control call codes 141
- SmartPort interface 139-142
- 'snd ' resource type 85

- software for product development 24-26
- sound. *See also* CD-Audio addressing 122 AppleCD SC and 75-103 in applications 78-83 Audio CD Access file 31-33 audio control calls 121-135, 177-198 controlling audio features 88 digitizers 86 hardware for capturing 86 HyperCard and 84-87 processor 76-80 sampling rates 76, 85 software for sound production 87 storing for HyperCard 85 stacks, HyperCard 51-52 standard Macintosh result codes 112 standards. *See* High Sierra; ISO 9660 StartUnit/StopUnit (\$18/\$19) calls 173 status calls, Macintosh CD-ROM 115-116, 136-138 DiskStatus (8) 136 GetSense (99) 138 GetSize (98) 137 WhoIsThere (97) 137 stop address 133, 185 SystemIdentifier field 40 SystemUse field 40-45

T

- table of contents of CD 127, 193
- tape premastering 20
- testing discs 23
- TestUnitReady (\$05) call 143

- Test Unit Ready command 143
- track number addressing format 121
- tracks 122

U

- unit attention page 149-150
- updating information 12
- updating server volume 107
- UserEject (80) call 120
- UserModify global property (HyperCard) 72
- user interface
 - general principles for 13, 15
 - HyperCard 59-70
 - SmartPort 139-142

V

- VarBlk (79) call 119
- Verify (\$1C) call 174
- Verify command 174
- visual cues and conventions xiv
- volume. *See also* file format or *specific format* naming 106 updating 107

W

- waveform, sound 79
- WhoIsThere (97) call 137

X

- XCMDs 58, 62

Y, Z

- Yellow Book* 29, 121

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word software. Proof and final pages were created on the Apple LaserWriter® printers. The final screen shot illustrations were created on the Linotronic L300 imagesetter. POSTSCRIPT®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type and display type are Apple's corporate font, a condensed version of Garamond. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.